UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Ingeniería Técnica en Informática de Gestión

PROYECTO FIN DE CARRERA

IMPLEMENTACIÓN DE UN PROGRAMA EN C# PARA LA CONSTRUCCIÓN Y EVALUACIÓN DE CONJUNTOS CLASIFICADORES BASADOS EN REDES DE NEURONAS

Autor: Mario Álvarez Zamorano

Tutor: M. Paz Sesmero Lorente

Octubre, 2011

Agradecimientos

Quiero expresar mi enorme gratitud hacia todas aquellas personas que estuvieron a mi lado durante todo el transcurso de mis años de carrera, todos aquellos que siempre me apoyaron, me animaron, me ayudaron y, sobre todo, me sufrieron.

Gracias a Lourdes, mi esposa, siempre, y especialmente en el tramo final de este proyecto. A mis hijos Candela y Mario, a quienes quiero dedicar este trabajo y que algún día lo recordemos cuando completen sus propias carreras. Gracias a mis padres, que me abrieron todas las puertas y me ofrecieron todas las oportunidades; y a mi tía Mª Carmen, que seguro se alegrará como nadie del éxito de mi proyecto. El agradecimiento y el recuerdo especial para todos aquellos que ya no están pero que se sentirían enormemente contentos y orgullosos de mí.

A todos los compañeros de clase, todos aquellos compañeros de prácticas, de estudio, y también de partida o de tenis. Gracias a Pedro, a Oscar, a Alfredo por tantos buenos momentos. Muchas gracias a mis amigos, sé que siempre están ahí por mucho tiempo que pase. Gracias a toda esa panda de incansables deportistas que seguro celebrarán este momento como si hubiéramos ganado el campeonato del mundo.

Muchas gracias a todos mis profesores, a los miembros del tribunal, y gracias, muchísimas gracias, a M Paz, mi seño, por su derroche de paciencia, apoyo, dedicación y esfuerzo sin los cuales este proyecto no hubiera sido posible.

Resumen

Un clasificador se define como un sistema artificial capaz de abstraer las características específicas de los objetos o datos que se le presentan y agruparlos en clases o categorías previamente aprendidas. En las últimas décadas, los estudios relativos a las tareas de clasificación se han visto impulsados por la idea de construir conjuntos de clasificadores. Es decir, sistemas en los que la tarea de clasificación se resuelve gracias a la intervención de una colección de clasificadores simples cuyas decisiones se combinan con el objetivo de construir un sistema más preciso que cualquiera de sus miembros.

En este trabajo se describen algunos de los métodos de generación de conjuntos más referenciados en la bibliografía y se presenta una aplicación con la que es posible construir, evaluar y comparar algunos de estos modelos. Ante un determinado dominio de aplicación, los resultados derivados de esta comparación servirán para estimar qué modelo es el más adecuado.

El funcionamiento y la validez de la aplicación desarrollada se han comprobado aplicándolo en un problema de reconocimiento y clasificación de señales de tráfico.

Palabras clave: Clasificador, Red de Neuronas Artificial, *Back-Propagation*, Conjunto de Clasificadores, *Bagging*, *ECOC*, Uno contra todos, F-Test, Test de McNemar

Abstract

A classifier is defined as an artificial system able to abstract objects' specific features or data received and group them in classes or categories previously learned. During the last decades, studies about classification issues have been driven by the idea of building groups of classifiers. This means that the task of classification is resolved by the intervention of a collection of simple classifiers whose decisions are combined with the aim of building a more accurate system than any of its members.

This paper describes some of the more popular methods for generating groups and presents an application that can build, evaluate and compare some of these models. In the presence of a specific application domain, results derived from this comparison will be used to estimate which model is the most appropriate.

The performance and validity of the developed application have been checked by its application in a traffic signal recognition and classification problem.

Keywords: Classifier, Artificial Neuron Network, BackPropagation, Ensemble of Classifiers, Bagging, ECOC, One Against All, F_Test, McNemar Test.

Índice General

Índice de Figuras

Índice de tablas

Índice de Ecuaciones

$$GD = 1 - \frac{p_2}{p_1}$$ [Ec. 2.7]

Medida de concordancia -*Interrater agreement*- 29

$$\kappa = 1 - \frac{\frac{1}{L}\sum_{k=1}^{N} l(x_k)(L - l(x_k))}{N(L-1)\overline{p}(1-\overline{p})}$$ [Ec. 2.8]

Diversidad de coincidencia de fallo, *CFD*. 29

$$CFD = \begin{cases} 0 & \text{si } p_0 = 1.0 \\ \frac{1}{1-p_o}\sum_{i=1}^{L}\frac{L-i}{L-1}p_i & \text{si } p_0 < 1 \end{cases}$$ [Ec. 2.9]

Cuenta de Borda 30

$$B(j) = \sum_{i=1}^{n} B_i(j)$$ [Ec. 2.10]

MAX / MIN (máximo / mínimo) 30

$$MCS_{MAX}(x) = \max\{C_1(x), ..., C_R(x)\}$$ [Ec. 2.11]

$$MCS_{MIN}(x) = \min\{C_1(x), ..., C_R(x)\}$$ [Ec. 2.12]

SUM (sumatorio) 30

$$MCS_{SUM}(x) = \sum_{j=1}^{R} C_j(x) w_j$$ [Ec. 2.13]

Función de activación (también conocida como función de excitación) 36

$$a_i = f\left(\sum_{j=1}^{M_i} w_{ij}x_{ij} + I\right) = f\left(\sum_{j=0}^{M_i} w_{ij}x_{ij}\right)$$ [Ec. 3.1]

Regla delta generalizada 39

$$\mathrm{E} = \frac{1}{N}\sum_{n=1}^{N} e(n)$$ [Ec. 3.2]

Error cometido por la red para el patrón *n* 39

$$\mathrm{e(n)} = \frac{1}{2}\sum_{i=1}^{n_c}(s_i(n) - y_i(n))^2$$ [Ec. 3.3]

Ajuste del peso 39

$$\Delta \mathrm{w}_{\mathrm{ij}}(\mathrm{t}) = \alpha \delta_{\mathrm{i}}(\mathrm{t}) \mathrm{S}_{\mathrm{j}}(\mathrm{t})$$ [Ec. 3.4]

$$f = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}(p_i^{(j)})^2}{2\sum_{i=1}^{5}s_j^2} \qquad \text{[Ec. 3.15]}$$

Cálculo McNemar 49

$$X^2 = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}} \qquad \text{[Ec. 3.16]}$$

Capítulo 1

Introducción y objetivos

1.1 Introducción

Las tareas de clasificación están presentes en una amplia cantidad de dominios como el diagnóstico médico: observados determinados síntomas se puede llegar a diagnosticar la enfermedad que sufre un paciente; la detección de fraudes: determinados valores en un análisis de cuentas pueden ser indicio de un posible fraude; la detección de errores en un proceso de fabricación: observando ciertas características de un producto final, se determina si durante el proceso de su fabricación pudo surgir algún problema; etc. La información generada en estos y otros dominios crece a tal ritmo que cada vez se hace más necesaria la construcción de sistemas que automaticen las tareas de clasificación.

Al igual que la mayor parte de las aplicaciones enmarcadas en *Inteligencia Artificial*, la creación de clasificadores nace con la pretensión de emular el comportamiento humano. En concreto, los clasificadores surgen del afán por crear sistemas capaces de asignar categorías o clases a los datos que se le presentan de forma análoga a como lo haría el ser humano. Aunque para la mayor parte de nosotros las tareas de clasificación son, en muchos casos, actos cotidianos que se realizan de forma inconsciente, su materialización no sería posible si previamente no se hubiera adquirido el conocimiento necesario para poder realizarlas. Este conocimiento se puede adquirir llevando a la práctica distintas formas de aprendizaje como deducción, analogía o memorización pero para muchos autores la forma más habitual de adquirirlo es *inducirlo* a partir de ejemplos o datos previamente clasificados. En base a lo anterior, un clasificador se define como un sistema artificial capaz de abstraer las características específicas de los objetos o datos que se le presentan y agruparlos en clases o categorías previamente aprendidas [Ranawana & Palade, 2005].

Somos conscientes de que nuestras decisiones mejoran en base a nuestro aprendizaje y a nuestra experiencia, y en muchas ocasiones tratamos de ser más precisos y de asegurarnos que una decisión es la acertada apoyándonos en las opiniones de nuestros amigos, compañeros o especialistas en una materia dada. La extrapolación de esta proposición al campo de los sistemas de clasificación deriva en la construcción de los denominados *conjuntos de clasificadores*. Es decir, en la construcción de un grupo de clasificadores cuyas decisiones individuales se combinan (típicamente mediante un sistema de votos que en ocasiones es ponderado) con el objetivo de obtener un clasificador más preciso que cualquiera de sus miembros [Dietterich, 1997].

Aunque, en las últimas décadas, y dentro del marco del *Aprendizaje Automático*, el estudio de los conjuntos de clasificadores es una de las áreas que más interés ha suscitado, la creación de un *clasificador universal*, es decir, un clasificador perfecto en todos los dominios y tareas, sigue siendo un problema abierto. El amplio abanico de modelos de clasificación disponible y la imposibilidad de conocer a prior cuál de ellos se ajustará mejor a las características especificas de un dominio dado, obligan a medir, comparar y estimar las bondades y defectos de cada uno de los modelos sobre el dominio considerado. Para ello, diversos autores [Dietterich, 1998; Kuncheva & Whitaker, 2002; Alpaydin, 1999] sugieren medidas y métodos estadísticos con los que estimar la opción que, en cada caso, parece más acertada.

1.2 Objetivos

Este Proyecto Fin de Carrera pretende dar una visión general de algunos de los métodos de generación de conjuntos de clasificadores referenciados en la Bibliografía y ser una herramienta para la construcción y evaluación de los mimos gracias al desarrollo de una aplicación que:

1. Provea de los métodos necesarios para construir conjuntos de clasificadores basados en redes de neuronas artificiales.
2. Implemente métodos estadísticos para comparar la precisión de los distintos modelos y ayuden a seleccionar la arquitectura más adecuada en el domino de estudio.
3. Ofrezca una interfaz sencilla que contenga y automatice los procesos de creación, aprendizaje, evaluación y comparación de los algoritmos desarrollados a partir de ficheros de datos basados en los formatos del proyecto de la Universidad de Stutgart *SNNS* (Stutgart Neural Network Simulator).

Este documento constituye la base teórica y el manual para entender la aplicación, con la estructura que se detalla a continuación.

El capítulo 2, Estado del Arte, desarrolla el concepto de conjunto de clasificadores, los tipos que existen, las fases requeridas en su construcción y cómo se evalúan frente a otros para determinar el mejor conjunto de clasificadores aplicable a un dominio concreto.

En el tercer capítulo se detalla el concepto de Red de Neuronas Artificial, se profundiza en las peculiaridades de los Conjunto de Clasificadores implementados y se describen los Métodos Estadísticos empleados para la comparación de los modelos de clasificación creados.

Después, en el capítulo 4, se muestra el trabajo experimental realizado, ofreciendo una descripción del sistema, resaltando las dificultades encontradas y las soluciones ofrecidas, así como los resultados empíricos obtenidos al aplicarlo a un problema de reconocimiento y clasificación de señales de tráfico.

El análisis general del trabajo realizado, las conclusiones derivadas y las pautas a seguir en el futuro se recogen en el capítulo 5, así como las dificultades encontradas en el transcurso del proyecto.

El Capítulo 6 incluye una planificación del desarrollo de la aplicación, compuesta por un diagrama *Gantt* de las fases del proyecto y un presupuesto del coste de su realización.

Por último, se pueden encontrar los capítulos de Referencias, un manual de usuario de la aplicación explicando gráficamente las distintas opciones de las que está compuesta y un manual de referencia para entender los aspectos más importantes del desarrollo de la aplicación.

Capítulo 2

Estado del Arte

El objetivo de este capítulo es brindar una visión general del área en la que se enmarca este trabajo. En primer lugar y a modo de introducción, se hace una breve descripción del concepto de *Aprendizaje Inductivo* definiendo de forma concisa los diferentes tipos que existe. Posteriormente en la sección 2.2 se desarrolla el concepto de *conjunto de clasificadores*. Posteriormente, en las secciones 2.3 y 2.4 se detallan las fases requeridas para construir un conjunto de clasificadores.

2.1. Introducción

El éxito a la hora de tomar cualquier decisión, por muy simple que sea, radica en buena parte en la percepción y el análisis de la información de la que se dispone. Un clasificador es un sistema capaz de diferenciar elementos en función de sus características y agruparlos en órdenes o clases. Esta tarea, que a priori parece sencilla, resulta inviable si el clasificador no cuenta con el conocimiento necesario para poder realizarla. Aunque es posible adquirir este conocimiento llevando a la práctica distintas formas de aprendizaje (deducción, analogía, memorización), la forma más habitual de adquirirlo es *inducirlo* a partir de ejemplos.

Las técnicas de aprendizaje inductivo se pueden dividir en dos grandes grupos: Aprendizaje Supervisado y Aprendizaje no Supervisado:

a) Aprendizaje Supervisado: Estas técnicas parten de un conjunto de ejemplos previamente etiquetados (datos de entrenamiento) y tiene como objetivo encontrar la hipótesis o función que determina la clase a la que pertenecen estos ejemplos y que a su vez permite predecir la categoría a la que pertenecen los nuevos datos que se puedan presentar. Dependiendo de la naturaleza de la etiqueta clase, se distinguen dos tipos de problemas: clasificación y regresión. Se habla de clasificación, cuando el conjunto de clases tiene una cardinalidad finita, es decir, cuando las clases toman valores discretos. Por el contrario, si los valores que puede tomar la etiqueta clase son continuos se habla de regresión.

.

b) Aprendizaje No Supervisado: En este caso, se parte de un conjunto de datos para los que se desconoce la clase a la que pertenecen y el objetivo es descubrir el esquema que permite organizar estos datos en clases. El resultado del proceso de aprendizaje son subconjuntos de datos caracterizados por contener patrones con características similares entre sí pero diferentes a las de los patrones contenidos en los otros subconjuntos.

El trabajo presentado en este informe, versa sobre el diseño de algoritmos que generan modelos de clasificación partiendo de un conjunto de datos etiquetados.

A la hora de diseñar un modelo de clasificación, se presentan dos limitaciones que impiden encontrar una función correcta para todos los casos que se puedan presentar. En primer lugar, la presencia de ruido en los datos de entrenamiento afectará a la calidad y el rendimiento del modelo de clasificación. Por otro lado, un insuficiente número de ejemplos suele ser un claro ejemplo de limitación en la mayoría de problemas de aprendizaje. Se ha observado experimentalmente que debido a estas limitaciones es prácticamente imposible construir, para cualquier entorno, un clasificador perfecto [Ranawana & Palade, 2005].

Con el objetivo de afrontar estas limitaciones, una gran parte de las investigaciones relativas a las tareas de clasificación se han centrado en construir sistemas en los que se combinan las decisiones dadas por un determinado conjunto de clasificadores. Estos sistemas reciben, entre otros, el nombre de Conjuntos de Clasificadores y se han convertido en una de las principales líneas de investigación de las últimas décadas.

2.2. Conjuntos de Clasificadores

Un conjunto de clasificadores (*ensemble of classifiers*) es un sistema de clasificación en el que la tarea de clasificación se resuelve haciendo uso de una colección de clasificadores cuyas decisiones individuales se combinan para obtener un clasificador más preciso que cualquiera de sus miembros [Dietterich, 2000].

Bajo el denominador de conjuntos de clasificadores se engloban un gran número de algoritmos: *Bagging* [Breiman, 1996], *Boosting* [Schapire, 1990], *ECOC* [Dietrerich & Bakiri, 1995], *Stacking* [Wolpert, 1992], etc. cuyo objetivo es la construcción de un clasificador robusto haciendo uso de clasificadores más simples denominados clasificadores base.

Los conjuntos de clasificadores tratan, en líneas generales, de reproducir el comportamiento humano a la hora de tomar una decisión de relativa importancia. Por ejemplo, un consejo de dirección de una empresa estará compuesto por profesionales con diferentes niveles de cualificación y experiencia, cuyas distintas interpretaciones de cada situación determinarán una decisión final consensuada con el fin de cerciorarse, en la medida de lo posible, de que la decisión tomada es la correcta.

La construcción de los conjuntos de clasificadores se divide en dos fases: Generación e Integración. El objetivo de la fase de generación es construir, a partir de los datos de entrenamiento, los clasificadores individuales (clasificadores base) que integrarán el conjunto. En la fase de integración, los resultados dados por los

clasificadores base se combinan para dar lugar a una hipótesis que pretende ser más precisa que cualquiera de las hipótesis resultantes de los clasificadores base.

A continuación se describen en detalle cada una de estas fases.

2.3. Fase de Generación.

Para que un conjunto de clasificadores mejore la precisión de cualquiera de los miembros que lo componen, se requiere que éstos sean precisos y diversos [Hansen & Salamon, 1990]. Un clasificador se dice que es preciso si el error cometido al clasificar nuevos ejemplos es menor que el que se cometería asignando una clase de forma aleatoria. Por otro lado, los clasificadores se consideran diversos si las decisiones erróneas se producen sobre ejemplos distintos.

Requerir que los clasificadores base sean precisos es una condición fácil de justificar y verificar. Partiendo de hipótesis imprecisas, difícilmente se conseguirá una hipótesis precisa. Además, una forma sencilla y bastante aceptada para determinar la precisión de un clasificador es calcular el porcentaje de ejemplos de test correctamente clasificados. Si el valor obtenido es mayor que el que se obtendría clasificando los ejemplos de forma aleatoria, el clasificador se puede considerar preciso.

Para clarificar la necesidad de requerir clasificadores base diversos se considerará un ejemplo en el que existen tres clasificadores, h_1, h_2 y h_3 y un ejemplo a clasificar, $\boldsymbol{x}$. Si los clasificadores no son diversos, es decir, cometen los mismos errores, y el ejemplo es clasificado erróneamente por uno de ellos, también será clasificado erróneamente por los otros dos. Por tanto, la decisión final del conjunto clasificador será errónea. Por el contrario, si los clasificadores base son diversos, cuando la predicción dada por h_1 sea errónea, las dadas por h_2 y h_3 serán correctas y, por tanto, si todas ellas tienen igual relevancia, al combinarlas, la predicción global del sistema será correcta.

En la Figura 2.1 se muestra un hipotético conjunto formado por 21 hipótesis, cada una de ellas con una tasa de error del 0.3. El área bajo la curva en donde 11 o más hipótesis estén simultáneamente erradas es 0.026, que es mucho menor que la tasa de error individual de las hipótesis [Dietterich, 1997].

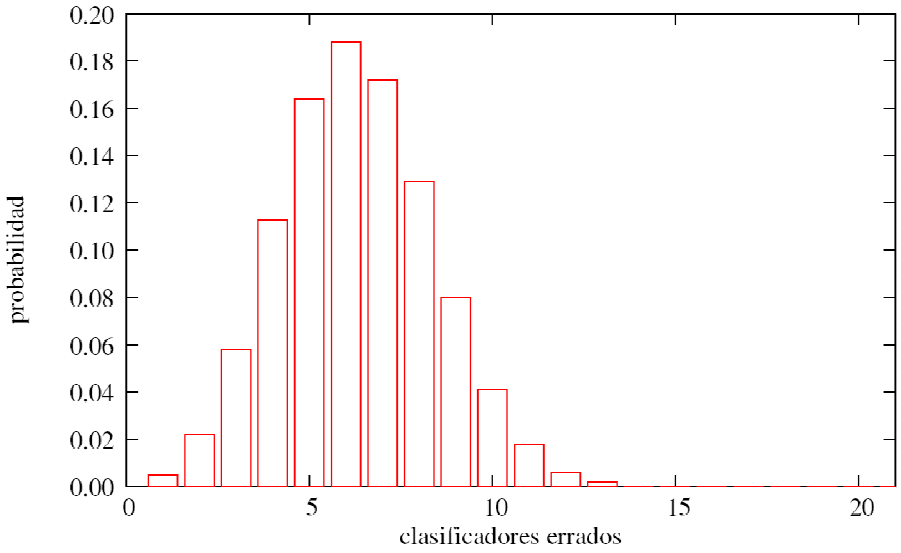


Figura 2.1: Probabilidad de que exactamente 1 (de 21) hipótesis comenta un error, asumiendo que cada hipótesis tiene una tasa de error de 0,3 y cometen sus errores independientemente de las demás hipótesis.

Aunque está demostrado que una condición necesaria para obtener un conjunto de clasificadores más preciso que los clasificadores base que lo integran es que éstos sean diversos, encontrar un grupo de clasificadores diversos con los que se garantice que el conjunto final es preciso no es una tarea trivial. La principal razón está en que, al contrario de lo que sucede con la precisión, los investigadores de este campo no han llegado a un consenso sobre cómo cuantificar esta magnitud ni cómo relacionarla con la precisión del conjunto. Es decir, hay varios parámetros que permiten estimar la diversidad de los clasificadores base pero la relación entre tales parámetros y la precisión del conjunto de clasificadores es aún una cuestión abierta [Kuncheva & Whitaker, 2003].

2.3.1. Generación de clasificadores diversos

Existen varias técnicas para generar clasificadores diversos, todas ellas basadas en el hecho de que la hipótesis asociada a un clasificador depende del algoritmo de aprendizaje y de los ejemplos usados en su construcción.

Las técnicas desarrolladas para generar diversidad variando el conjunto de datos de entrenamiento pueden ser clasificadas en tres grandes grupos [Dietterich, 1997]: manipulación del conjunto de entrenamiento, manipulación de los atributos de entrada y manipulación de las salidas. A continuación se detallan estas técnicas:

- *Manipulación del conjunto de entrenamiento*. Una forma de construir clasificadores diversos es utilizar un subconjunto de datos de entrenamiento distinto en la fase de generación de los clasificadores base. Si los ejemplos usados en la construcción del clasificador son diferentes, las hipótesis obtenidas serán también diferentes.

Dentro de las técnicas que manipulan el conjunto de entrenamiento con el objetivo de generar clasificadores diversos destacan *Bagging* [Breiman, 1996], los *comités de validación cruzada* (en inglés, *cross-validated committees*) [Parmanto et al., 1996] y *Boosting* [Schapire 1990].

Así*, Bagging*, a partir del conjunto de ejemplos original genera un grupo de subconjuntos en los se mantiene la cardinalidad inicial pero en los que, con respecto al conjunto original, algunas instancias están repetidas y otras omitidas.

Otra forma de generar distintos subconjuntos de entrenamiento es dividir el conjunto de datos inicial en *K* particiones disjuntas de igual tamaño y, posteriormente, generar *K* subconjuntos distintos compuestos, cada uno de ellos, por todos los elementos contenidos en *K-1* de estas particiones. Este proceso coincide con el usado en validación cruzada por lo que los métodos que lo aplican se denominan *comités de validación cruzada*.

El método *Boosting*, propuesto por [Freund & Schapire, 1996], se basa en la ponderación de cada ejemplo del conjunto de datos de entrenamiento. En cada iteración, el peso de los ejemplos varía, incrementándose en los casos en los que el ejemplo ha sido erróneamente clasificado y reduciéndose en los ejemplos acertados [Hernández-Orallo et al., 2007].

- *Manipulación de los atributos de entrada*. Los atributos que describen los ejemplos son modificados cuantitativa y/o cualitativamente. La modificación cuantitativa permite reducir la cantidad de atributos en cada uno de los subconjuntos de entrenamiento por medio de distintos métodos: por selección aleatoria [Bryll et al., 2003], aplicando distintos algoritmos de selección de características [Blum & Langley, 1997] o aplicando conocimiento dependiente del dominio (p.e., agrupando los atributos en función de la fuente de la que proceden). Las modificaciones cualitativas están vinculadas a procesos de inducción constructiva [Zeng, 1996] y, por tanto, conllevan un cambio en el espacio de representación de los atributos que describen los ejemplos (cambio en los valores que pueden tomar o generación de nuevos atributos).

- *Manipulación de las salidas*. Una tercera técnica para generar clasificadores diversos es modificar la clase asociada a los ejemplos. Estos métodos tienen su mayor utilidad cuando los datos de entrenamiento están asociados a un número grande de clases. El procedimiento más habitual consiste en descomponer el problema multiclase en subproblemas binarios, resolver estos problemas y combinar las soluciones encontradas con el objetivo de dar solución al problema de partida.

  En función del esquema usado en la descomposición del problema, estos sistemas se pueden subdividir en las siguientes categorías [Ou & Murphey, 2007]:

  - Sistemas basados en una arquitectura *Uno Contra Todos (OAA).* En esta arquitectura cada clasificador binario se especializa en separar los ejemplos pertenecientes a una clase respecto de los ejemplos pertenecientes a cualquiera de las clases restantes.

- Sistemas basados en una arquitectura *Uno Contra Uno (OAO).* En este caso, cada clasificador binario tiene como objetivo distinguir los ejemplos de una determinada clase respecto de los pertenecientes a otra clase dada.
- Sistemas basados en una arquitectura *p contra q (PAQ).* Al aplicar esta metodología, cada clasificador binario se especializa en distinguir los ejemplos asociados a *P* clases respecto de los asociados a *Q* clases (*P, Q ≥1; P+Q≤Nº total del clases*).

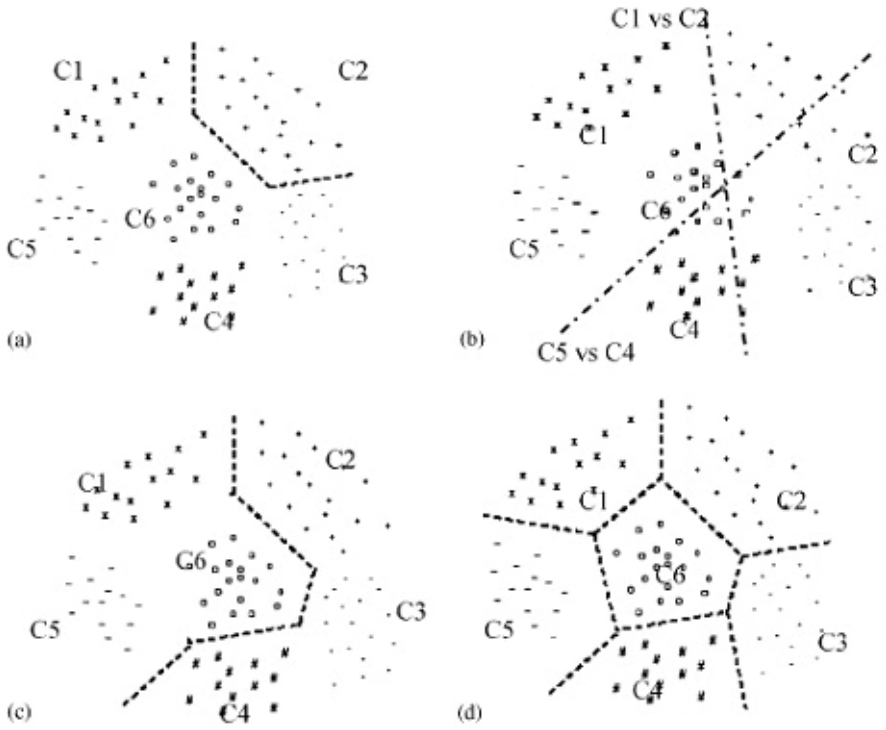


Figura 2.2. Ilustración de varios límites de clasificación generados empleando distintos métodos de entrenamiento para clasificación de patrones de K-clases (a) Límite de clasificación generado por una red de neuronas entrenada mediante el método Uno-contra-todos (OAA); (b) Dos límites de clasificación generados por dos redes de neuronas entrenadas con Uno-contra-uno (OAO); (c) Límite de clasificación generado por una red de neuronas entrenada con el método PAQ; y (d) límite de clasificación optimo que separa las seis clases en el entorno de las características.

Un método representativo de las técnicas basadas en la manipulación de las salidas y, en particular, de la arquitectura p contra q es ECOC (*Error correcting output code*) [Dietterich & Bakiri, 1995]. En este método los ejemplos de entrada son re-etiquetados como pertenecientes a dos clases, es decir, el conjunto de clases inicial $C=\{c_1, c_2,... c_k\}$ es dividido aleatoriamente en dos subconjuntos C_l+ y C_l- de forma que los ejemplos asociados a las clases contenidas en C_l^+ son re-etiquetadas con 1 y las instancias pertenecientes a cualquier clase contenida en C_l^- son re-etiquetadas con 0. Al repetir este proceso *L* veces se obtienen *L* conjuntos de entrenamiento distintos y, en consecuencia, *L* clasificadores distintos.

Una vez creados todos los clasificadores base, la reconstrucción del problema inicial implica admitir que una instancia clasificada con un 1 por el *i-ésimo* clasificador podrá pertenecer a cualquiera de las clases contenidas en C_i^+ y, por tanto, cada una de ellas recibirá un voto. Por el contrario, si la salida del *i-ésimo* clasificador es 0, la instancia podrá pertenecer a cualquiera de las clases contenidas en C_i^-, por lo que cada clase en C_i^- recibirá un voto. Una vez que se conoce la decisión dada por los *L* clasificadores, la etiqueta que se le asigna a la instancia dependerá del número de votos que ha recibido cada una de las clases.

Junto a las técnicas basadas en la manipulación de ejemplos, se encuentran los llamados métodos de generación de conjuntos heterogéneos, en los cuales el conjunto de clasificadores es generado por varios algoritmos de aprendizaje distintos. El más conocido es el llamado método *Stacking* [Wolpert, 1992], que utiliza diferentes algoritmos de aprendizaje para generar el conjunto de clasificadores (árboles de decisión, redes de neuronas, etc.). Tras generar los clasificadores, *Stacking* utiliza el llamado meta-clasificador (o modelo de nivel-1) que actúa sobre las decisiones de los clasificadores base (o modelos de nivel-0) para obtener una salida final del conjunto.

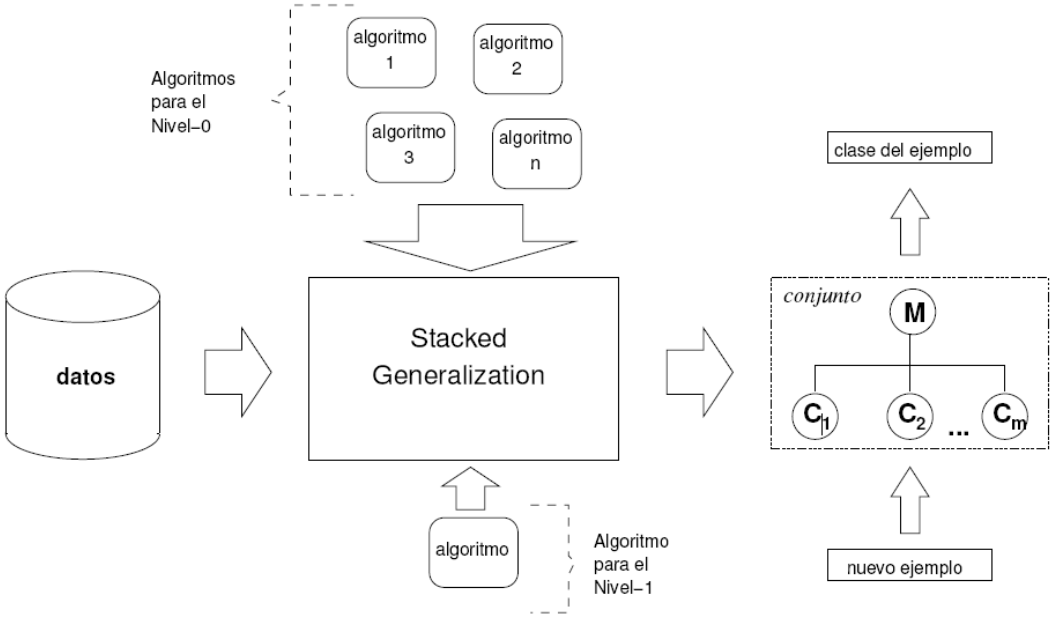


Figura 2.3. Esquema de funcionamiento general del método Stacking

Dado un conjunto de datos *S*, *Stacking* genera, en primer lugar, un subgrupo de conjuntos de entrenamiento S_1, ..., S_T para luego seguir un proceso similar al de la validación cruzada: se deja uno de los subconjuntos fuera (e.g. S_j) para utilizarlo posteriormente. El resto de las instancias $S(-j) = S-S_j$ son utilizadas para generar los clasificadores de nivel-0 mediante la aplicación de *K* algoritmos de aprendizaje distintos. Después de que los modelos de nivel-0 han sido generados, el conjunto S_j es utilizado para entrenar el meta-clasificador (clasificador de nivel-1). Los datos de entrenamiento de nivel-1 se forman a partir de las predicciones de los modelos de nivel-0 sobre las instancias en S_j, las cuales han sido reservadas para este propósito. Los datos de nivel-1 tienen *K* atributos cuyos valores son las predicciones de cada uno de los K

clasificadores de nivel-0 para cada instancia en S_j. De este modo, una instancia de entrenamiento de nivel-1 está constituida por K atributos (las K predicciones) y la clase objetivo. Una vez que los datos de nivel-1 han sido construidos a partir de todas la instancias en S_j, cualquier algoritmo de aprendizaje puede ser utilizado para generar el modelo de nivel-1. Para completar el proceso, los modelos de nivel-0 son regenerados a partir del conjunto S completo (de esta manera se espera que los clasificadores sean ligeramente mías precisos). Para clasificar una nueva instancia, los modelos de nivel-0 producen un vector de predicciones que es la entrada al modelo de nivel-1, el cual genera la predicción final del conjunto.

2.3.2. Medidas de diversidad

Como ya se ha señalado, una de las condiciones necesarias para obtener un buen conjunto de clasificadores es que los clasificadores base que lo integran sean diversos. No obstante, y a pesar de su relevancia, a día de hoy no existe una definición formal de este parámetro y, en consecuencia, no existe una expresión estándar con la que cuantificarla. Sin embargo, en la literatura, es posible encontrar medidas estadísticas y matemáticas cuyo valor se puede considerar un indicativo del grado de diversidad.
Dado que determinar el grado de diversidad entre dos clasificadores es intuitivamente más sencillo que estimar el grado de diversidad asociado a un grupo de clasificadores, estas medidas se suelen catalogar en dos grupos: medidas duales (*pairwise*) y medidas no duales (*no pairwise*).

Medidas duales. Estas medidas intentan establecer la diversidad existente entre las decisiones asociadas a dos clasificadores base. De ahí que cuando el conjunto está formado por 3 o más clasificadores base la estimación de la diversidad esté dada por el promedio de las medidas sobre todos los pares de clasificadores base.

Admitiendo que dos clasificadores se consideran diversos si las decisiones erróneas se producen sobre ejemplos distintos, parece claro que el grado de diversidad entre dos clasificadores, C_i y C_j, ha de ser función de:

N^{11}: Número de ejemplos clasificados correctamente por ambos clasificadores.
N^{10}: Número de ejemplos clasificados correctamente por el clasificador C_i pero erróneamente por el clasificador C_j
N^{01}: Número de ejemplos clasificados correctamente por el clasificador C_j pero erróneamente por el clasificador C_i
N^{00}: Número de ejemplos clasificados erróneamente por ambos clasificadores.

A continuación, y apoyándose en esta nomenclatura, se enumeran y definen matemáticamente las medidas duales recogidas en [Kuncheva & Whitaker. 2003]:

- **Estadístico Q**: Este estadístico, definido como:

$$Q_{ik} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \qquad \text{[Ec. 2.1]}$$

adquiere valores positivos cuando los clasificadores tienden a reconocer correctamente los mismos ejemplos y adquiere valores negativos cuando no existe correlación entre los errores cometidos por uno y otro clasificador:

- **Coeficiente de correlación, ρ** : Otra forma de expresar cuantitativamente la relación entre los aciertos y los errores cometidos por dos clasificadores es mediante el coeficiente de correlación, ρ_{ij}, dado por:

$$\rho_{ij} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{\left(N^{11}+N^{10}\right)\left(N^{01}+N^{00}\right)\left(N^{11}+N^{01}\right)\left(N^{10}+N^{00}\right)}} \qquad \text{[Ec. 2.2]}$$

- **Medida de desacuerdo, *Dis***: Esta medida se corresponde con el cociente entre el número de ejemplos clasificados correctamente por sólo uno de los clasificadores y el número total de ejemplos:

$$Dis_{ij} = \frac{N^{01}+N^{10}}{N^{11}+N^{00}+N^{01}+N^{10}} \qquad \text{[Ec. 2.3]}$$

- **Medida de doble fallo, *DF***: Dados dos clasificadores, la medida de doble fallo se define como el cociente entre los ejemplos clasificados erróneamente por ambos clasificadores y el número total de ejemplos que integran el conjunto de entrenamiento.

$$DF = \frac{N^{00}}{N^{11}+N^{00}+N^{01}+N^{10}} \qquad \text{[Ec. 2.4]}$$

<u>*Medidas no duales*</u>. El objetivo de estas medidas es estimar la precisión del conjunto considerándolo como un todo. En esta categoría se incluyen, entre otras:

- **Medida de Entropía, *E***:

$$E = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{(L-[L/2])}\min\left\{l(x_k), L-l(x_k)\right\} \qquad \text{[Ec. 2.5]}$$

donde:

l(xj): Número de clasificadores que reconocen correctamente el ejemplo x_j.
L: Número de clasificadores base que integran el conjunto
N: Número de ejemplos de entrenamiento

- **Varianza de Kohavi-Wolpert, *KW***

$$KW = \frac{1}{NL^2}\sum_{k=1}^{N} l(x_k)[L-l(x_k)] = \frac{L-1}{2L}Dis_{av} \qquad \text{[Ec. 2.6]}$$

- **Diversidad generalizada, *GD***

$$GD = 1 - \frac{p_2}{p_1} \qquad \text{[Ec. 2.7]}$$

donde:

p_1: es la probabilidad de que un clasificador seleccionado aleatoriamente clasifique erróneamente un ejemplo seleccionado al azar

p_2: es la probabilidad de que dos clasificadores seleccionados al azar clasifiquen erróneamente un ejemplo seleccionado aleatoriamente.

- **Medida de concordancia, κ.**

$$\kappa = 1 - \frac{\frac{1}{L}\sum_{k=1}^{N} l(x_k)(L - l(x_k))}{N(L-1)\bar{p}(1-\bar{p})} \qquad \text{[Ec. 2.8]}$$

donde:

$\bar{p}$ es la media de las precisiones de los clasificadores individuales.

- **Diversidad de coincidencia de fallo, *CFD***.

$$CFD = \begin{cases} 0 & \text{si } p_0 = 1.0 \\ \frac{1}{1-p_o}\sum_{i=1}^{L}\frac{L-i}{L-1}p_i & \text{si } p_0 < 1 \end{cases} \qquad \text{[Ec. 2.9]}$$

donde:

p_i es la probabilidad de que exactamente i clasificadores sean incorrectos, y p_0 es la probabilidad de todos los clasificadores sean correctos.

2.4. Fase de integración

La construcción de un conjunto de clasificadores implica generar una serie de clasificadores base diversos y precisos y, además, seleccionar un método de integración que combine las hipótesis asociadas a cada clasificador base y genere una hipótesis más precisa que cualquiera de las hipótesis individuales.

En función de su arquitectura, los esquemas usados en la fase de integración se pueden clasificar en las siguientes tres grandes categorías [Jain et al., 2000]:

2.4.1. **Arquitectura paralela**

Los clasificadores base son independientes unos de otros por lo que, ante un patrón de entrada todos emiten una decisión. Según la política usada para combinar estas decisiones y conseguir una única decisión final, las estrategias de combinación se pueden agrupar en las siguientes:

- Técnicas de fusión: En este caso, todos los clasificadores base intervienen en la decisión final del sistema. En la mayoría de los casos, esta decisión se obtiene combinando matemáticamente las salidas de los componentes individuales. Las expresiones matemáticas más usadas son:
  - Voto Mayoritario Simple o Moda. Cada clasificador base determina la clase que, a su juicio, está asociada al ejemplo. La clasificación más popular es la que resulta elegida por el conjunto como decisión final.
  - Voto Mayoritario Ponderado. Cada clasificador puede tener diferente grado de influencia en el resultado final. Un sistema para asignar pesos consistiría en entrenar a cada clasificador individualmente y evaluar después unos datos de prueba fijados. El peso del voto de cada clasificador se determinaría entonces por el resultado de la anterior evaluación.
  - Cuenta de Borda. Este método utiliza decisiones individuales por orden de preferencia sobre las clases. La cuenta de Borda individual de un clasificador *Ci* para la clase c_j , *Bi(j)*, es el número de clases con menor preferencia que c_j. La cuenta de Borda colectiva sobre la clase c_j. se evalúa a partir de las cuentas individuales de la siguiente manera:

$$B(j) = \sum_{i=1}^{n} B_i(j) \qquad \text{[Ec. 2.10]}$$

    La clase asignada por el conjunto se corresponde con la de mayor cuenta de Borda. Así, para un ejemplo de cuatro clases (a, b, c y d) y tres clasificadores en los que las salidas asignadas por orden de preferencia son: C_1=(a, d, b, c), C_2=(c, d, b, a) y C_3=(d, c, a, b), la cuentas de Borda colectivas para cada clase serán: *B(a)*=3+0+1=4, *B(b)*=1+1+0, *B(c)*=0+3+2=5 y *B(d)*=2+2+3=7. En este ejemplo, la clase con mayor cuenta de Borda es la *d*, que será la clase asignada por el conjunto.
  - MAX / MIN (máximo / mínimo) – son las funciones más simples, y consisten en seleccionar como salida del conjunto la de mayor/menor valor

$$MCS_{MAX}(x) = \max\{C_1(x), ..., C_R(x)\} \qquad \text{[Ec. 2.11]}$$

$$MCS_{MIN}(x) = \min\{C_1(x), ..., C_R(x)\} \qquad \text{[Ec. 2.12]}$$

  - SUM (sumatorio) – La salida final es el sumatorio de las salidas de cada clasificador base, o en una versión más avanzada, la suma de las salidas multiplicadas por un peso asignado a cada clasificador

$$MCS_{SUM}(x) = \sum_{j=1}^{R} C_j(x) w_j \qquad \text{[Ec. 2.13]}$$

En la Figura 2.4 se muestra un esquema de la integración en paralelo de los clasificadores base.

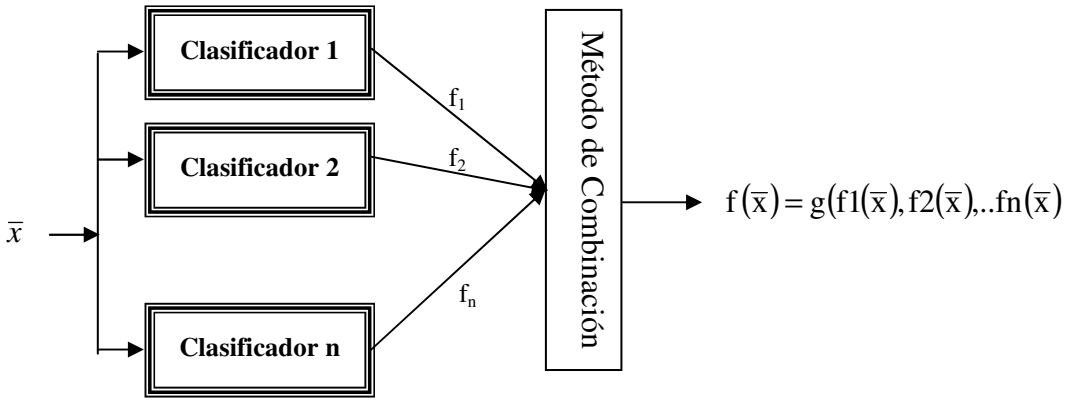


Figura 2.4. Conjunto de clasificadores integrados en paralelo. En este caso, la salida del sistema se determina combinando las salidas generadas por los clasificadores base.

- Técnicas de selección: Estas técnicas asumen que cada clasificador base es un *experto* en una determinada región del espacio [Zhu et al., 2004]. Por tanto, a la hora de clasificar una nueva instancia sólo se considera la decisión emitida por un único clasificador. Dependiendo de si la región de competencia de un clasificador se define durante la fase de entrenamiento o durante la fase de clasificación, las técnicas de selección se dividen en estáticas o dinámicas [Kuncheva, 2002].

2.4.2. **Arquitectura en cascada**

Los clasificadores base se invocan secuencialmente hasta que el patrón de entrada se considera clasificado. Por tanto, si la decisión emitida por uno de los clasificadores se considera fiable, el proceso de clasificación se detiene y la salida del conjunto coincide con la emitida por este clasificador. En caso contrario, es decir, si tras la llamada a un clasificador éste no puede determinar la clase a la que pertenece el patrón de entrada o su decisión no goza del suficiente crédito, dicho patrón será enviado al siguiente clasificador para que éste emita su decisión.

Un ejemplo representativo de esta arquitectura es la metodología OAHO (*One-Against-Higher-Order*) [Ou & Murphey, 2007; Murphey et al., 2007] usada para resolver problemas multiclase cuando los datos de entrenamiento no están balanceados. En la Figura 2.5 se muestra un esquema representativo de la arquitectura en cascada.

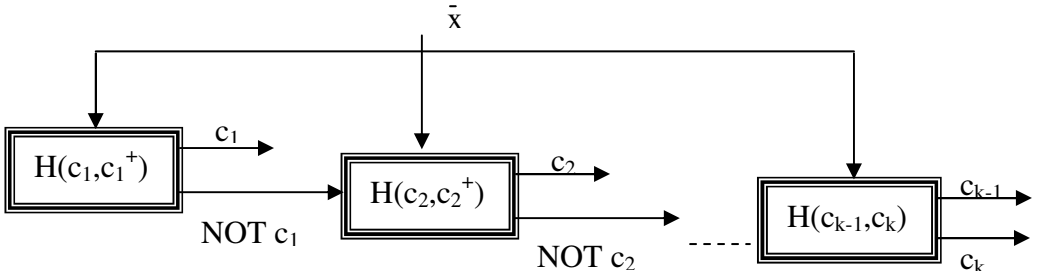


Figura 2.5 Conjunto de clasificadores integrados en paralelo que resuelve un problema multiclase aplicando la metodología OAHO.

2.4.3. **Arquitectura jerárquica**

Los clasificadores se organizan en una estructura con forma de árbol, en la que la llamada a un clasificador base depende de la salida dada por los clasificadores que, jerárquicamente, le preceden. La idea que subyace a esta arquitectura es que una tarea de clasificación relativamente compleja puede ser reemplazada por una combinación de clasificaciones más simples en la que la viabilidad de una clasificación está supeditada a los resultados de las clasificaciones anteriores.

La figura 2.6 muestra un esquema de esta arquitectura.

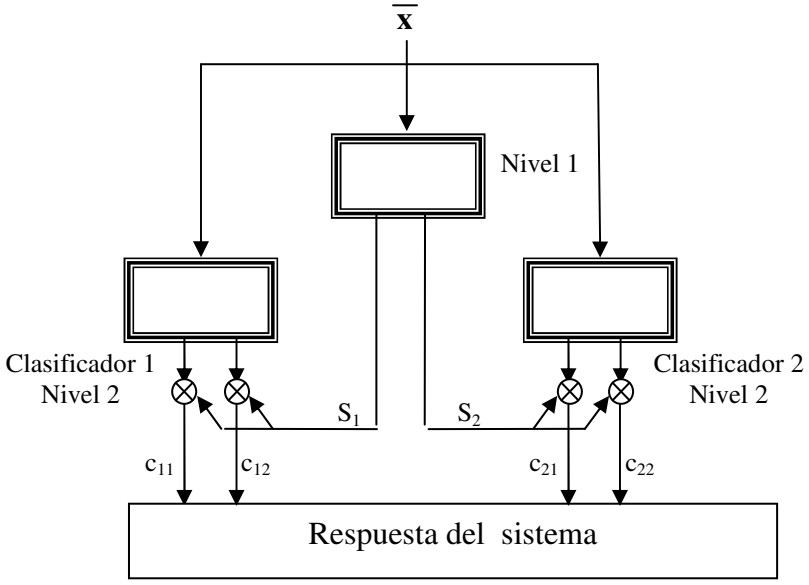


Figura 2.6. Conjunto de clasificadores integrados jerárquicamente usado en la resolución de un problema de 4 clases. El primer nivel está formado por un clasificador de 2 salidas y el segundo nivel por dos clasificadores con 2 salidas cada uno. La respuesta del sistema se obtiene multiplicando las salidas de ambos niveles. En este esquema el conjunto de clases está dividido en dos subconjuntos ($S_1=\{c_{11},c_{12}\}$, $S_2=\{c_{21},c_{22}\}$) cada uno de los cuales contiene 2 clases.

Capítulo 3

Descripción del Sistema

El objetivo de este Proyecto Fin de Carrera es la construcción de un programa que integra y evalúa distintos sistemas de clasificación. Una vez definido qué es un clasificador y desarrollado el concepto de conjunto de clasificadores, en este capítulo se detallan las características de los distintos modelos implementados y las medidas usadas para evaluar y comparar estos modelos.

Dado que todos los sistemas implementados siguen un modelo basado en Redes de Neuronas Artificiales, el presente capítulo comienza introduciendo el concepto de Red de Neuronas Artificial, cuáles son sus características y el Algoritmo de Aprendizaje Automático implementado en este proyecto para dotar a las Redes del conocimiento necesario.

3.1. Red de Neuronas Artificiales

Las Redes de Neuronas Artificiales consisten en una simulación de las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales. Una red de neuronas está compuesta por un conjunto de unidades de procesamiento llamadas neuronas.

Como se puede observar en la Figura 3.1, cada neurona recibe una serie de entradas a través de interconexiones ponderadas y emite una salida. Esta salida viene dada por dos funciones:

- Función de activación (también conocida como función de excitación), que, como su nombre indica, determina el valor de activación de la neurona a partir de las señales que recibe. Es decir:

$$a_i = f\left(\sum_{j=1}^{M_i} w_{ij}x_{ij} + I\right) = f\left(\sum_{j=0}^{M_i} w_{ij}x_{ij}\right) \qquad \text{[Ec. 3.1]}$$

  donde:
  x_{ij} representa las entradas de la neurona i
  w_{ij} son los pesos de las distintas entradas
  I umbral de la neurona (también denominado entrada interna o *bias*)

- Función de transferencia o salida. Cada neurona posee una función que transforma el estado actual de activación en una señal de salida. En la mayoría de los casos, esta función se corresponde con la función identidad.

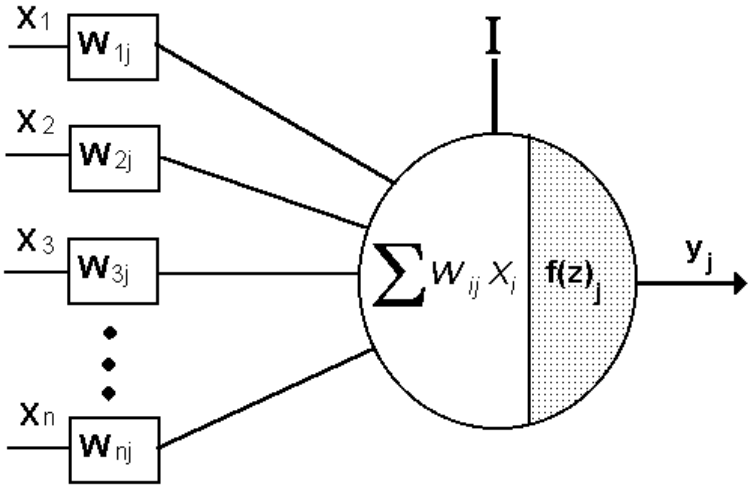


Figura 3.1. Esquema de funcionamiento de una neurona

3.1.1. Algoritmo de Aprendizaje

La base del aprendizaje en una Red de Neuronas Artificial radica en el ajuste de los pesos de sus conexiones. Por lo que se puede definir como aprendizaje el proceso por el cual los parámetros libres de una Red de Neuronas son ajustados a través de un proceso continuo de estimulación por parte del entorno en donde se sitúa el sistema. El tipo de aprendizaje viene determinado por la forma en la que tienen lugar dichos cambios.

En el aprendizaje supervisado, el ajuste de los pesos se calcula a partir de la comparación directa de la salida proporcionada por la red con la salida deseada (la proporcionada por los ejemplos). La diferencia entre la salida dada por la red y la salida esperada, condiciona la modificación de pesos.

3.1.2. Perceptrón Multicapa

Una de las clases de Redes de **N**euronas más usada en el aprendizaje supervisado es el *Perceptrón Multicapa*. Este modelo es una generalización del Perceptrón simple [Rosenblatt, 1959] y surge como consecuencia de las limitaciones de este último al problema de la separabilidad no lineal [Isasi & Galván, 2004].

El *Perceptrón Multicapa* es una Red de Neuronas caracterizada porque sus neuronas se agrupan en capas en distintos niveles. Como se puede ver en la Figura 3.2, la red consta de una o varias unidades de entrada, una capa de unidades de salida y una o varias capas de neuronas ocultas entre ellas, de tal manera que cada una de las neuronas de una determinada capa está conectada a todas y cada una de las neuronas de la capa siguiente.

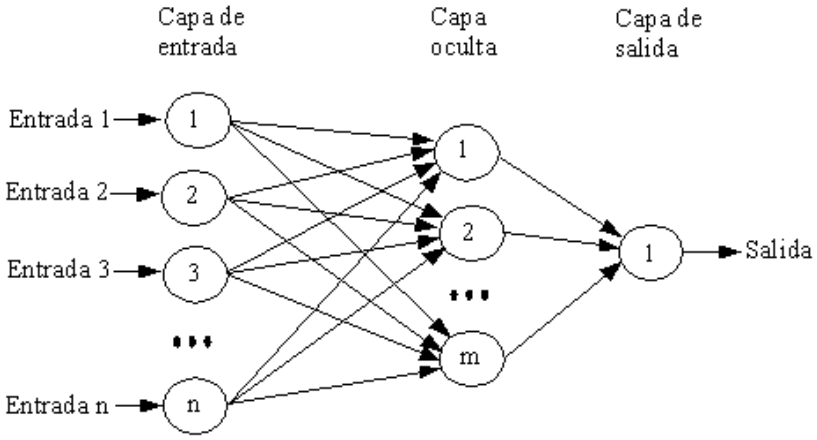


Figura 3.2. Estructura de un Perceptrón Multicapa con una sola capa oculta

La capa de entrada es un mero transmisor de las entradas o patrones a la primera de las capas ocultas. En estas capas ocultas es donde se lleva a cabo el procesamiento no lineal de los patrones recibidos pasando finalmente a la capa de salida, que proporciona la respuesta de la red al exterior. Las neuronas de una red, generalmente, están unidas a las neuronas del nivel posterior mediante una conexión ponderada dirigida siempre hacia delante. Cuando todas las neuronas de cada capa están conectadas a todas las neuronas de la capa siguiente, se dice que la red está totalmente conectada.

3.2. Algoritmo de aprendizaje: Regla de Retropropagación

Desde el punto de vista de las redes de neuronas, el objetivo del proceso de aprendizaje supervisado es conseguir que distintos ejemplos, es decir, distintos patrones de activación, generen una misma respuesta. Para ello, la red se somete a un proceso de aprendizaje en el que de forma secuencial se presentan a la red distintos ejemplos de los que se conoce la clase a la que pertenece. Si la respuesta generada coincide con la esperada, no sucede nada. En caso contrario, se modifican los pesos de las interconexiones hasta hacer coincidir la repuesta del sistema con la esperada.

Para conseguir que el sistema encuentre el conjunto de pesos con el que es posible conseguir la respuesta deseada para todos y cada uno de los patrones de entrada presentados, es necesario que las modificaciones realizadas tengan en cuenta no sólo la situación actual del sistema sino, también su historia pasada. Sólo así se garantiza que el afán por clasificar correctamente un ejemplo dado no repercute en exceso sobre los anteriores ajustes realizados.

Uno de los algoritmos de aprendizaje supervisado más usado en el entrenamiento del Perceptrón Multicapa es la *regla de retro-propagación (del inglés Back-Propagatión).* Esta regla, conocida también como regla delta generalizada, equivale a un algoritmo matemático cuyo objetivo es minimizar el error cuadrático del sistema, i.e., la función:

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n) \qquad \text{[Ec. 3.2]}$$

donde *N* es el número de patrones y *e(n)* es el error cometido por la red para el patrón *n*, dado por:

$$e(n) = \frac{1}{2}\sum_{i=1}^{n_c} (s_i(n) - y_i(n))^2 \qquad \text{[Ec. 3.3]}$$

siendo $Y(n)=(y_1(n), ..., y_{nc}(n))$ y $S(n)= (s_1(n), ..., s_{nc}(n))$, los vectores de salidas de la red y salidas deseadas para el patrón *n*, respectivamente.

De acuerdo con la regla delta generalizada, el peso w_{ij} que conecta las neuronas *i* y *j*, se ajusta por:

$$\Delta w_{ij}(t) = \alpha \delta_i(t) S_j(t) \qquad \text{[Ec. 3.4]}$$

donde:

α es el coeficiente de aprendizaje y su valor está incluido en el intervalo (0,1),
$\delta_i(t)$ es el error de la neurona i en el instante t, cuyo valor está dado por:

- Si se trata de una neurona de la capa de salida:

$$\delta_i(t) = (y_i(t) - s_i(t))\varphi'(t) \qquad \text{[Ec. 3.5]}$$

- Si se trata de una neurona de las capas ocultas:

$$\delta_i(t) = \varphi'(t)\sum_{h=1}^{k} (\delta_h(t) w_{hi}) \qquad \text{[Ec. 3.6]}$$

siendo:
φ' la derivada de la función que determina el valor de activación que la neurona *i*,
h el número de unidades que componen la capa destino.

Como función de activación *f*, en un Perceptrón Multicapa generalmente se utilizan las funciones siguientes:

- Función sigmoidal

$$f_1(x) = \frac{1}{1+e^{-x}} \qquad \text{[Ec. 3.7]}$$

- Función tangente hiperbólica

$$f_2(x) = \frac{1-e^{-x}}{1+e^{-x}} \qquad \text{[Ec. 3.8]}$$

Ambas son funciones crecientes don dos niveles de saturación, relacionadas mediante la expresión $f_2 = 2f_1(x) - 1$, por lo que la utilización de una u otra se elige únicamente en función del recorrido que interese [Isasi & Galván, 2004].

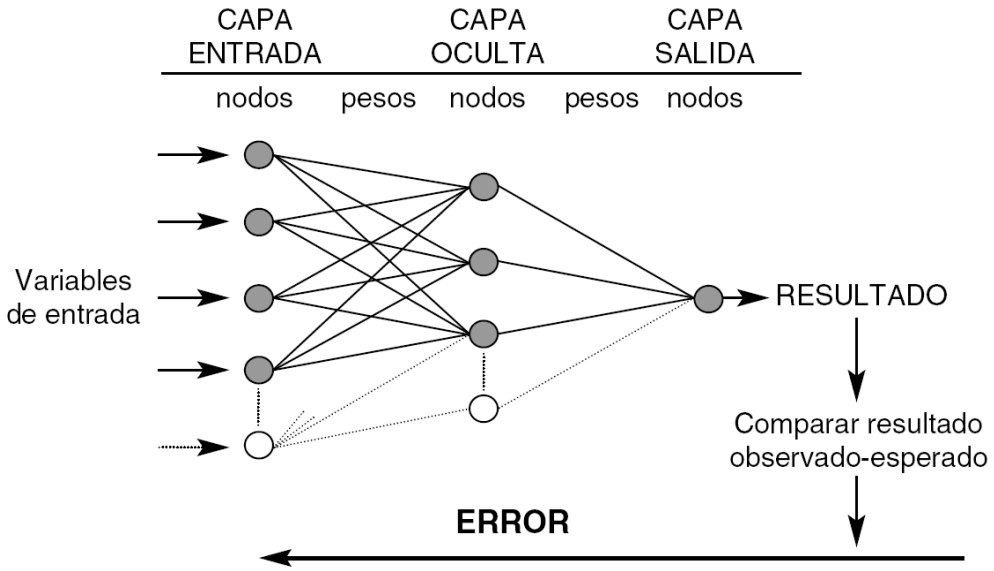


Figura 3.3. Esquema de funcionamiento del algoritmo de aprendizaje de RetroPropagación

Una vez descritas las características arquitectónicas y funcionales del Perceptrón Multicapa, a continuación se describen los conjuntos de clasificadores implementados en este proyecto. Para facilitar la terminología, en lo que sigue, se considerará que la una Red de Neuronas Simple es un conjunto de clasificadores compuesto por un único clasificador base y en el que la función de combinación se corresponde con la función identidad.

3.3. Bagging

Bagging (***B****ootstrapping and* ***agg****regat****ing***) es un sistema propuesto por Breiman, [Breiman, 1996] en el que cada clasificador base se genera a partir de un conjunto de datos obtenido del conjunto de entrenamiento original por muestreo aleatorio con reemplazamiento.

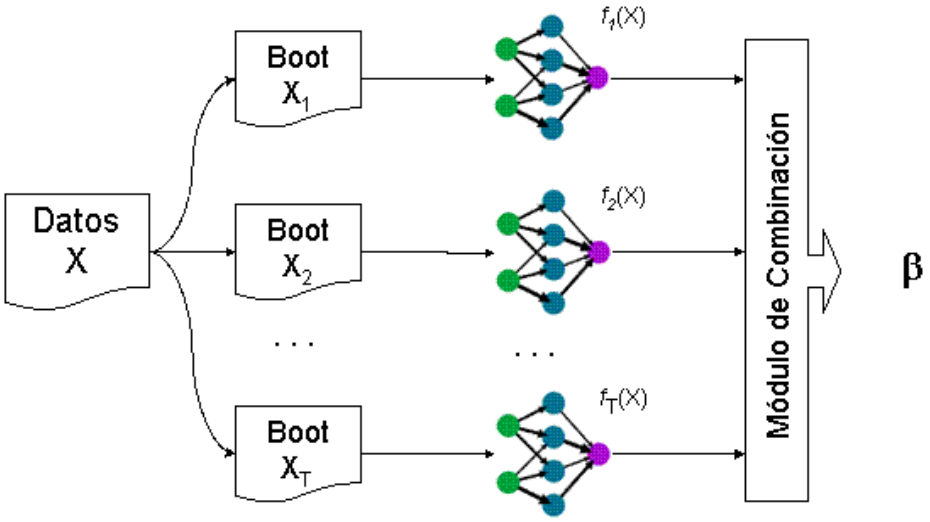


Figura 3.4. Diagrama de un sistema Bagging

El algoritmo de Bagging se puede resumir del siguiente modo:

Entradas:

Conjunto de entrenamiento, X
Algoritmo de aprendizaje base, B
Número de muestras (bootstrap) T

Procedimiento:

Para b=1 hasta T {
Generar un nuevo conjunto de datos por muestreo aleatorio, X_b
Construir un clasificador base, $f_b(x)$, a partr de X_b
}
Combinar los clasificadores base $f(x)$, {b=1,2,...B} usando como método integración el voto mayoritario:

$$\beta(x) = \arg\max_{C_i(x)=f(x)} \sum C_i(x) \qquad \text{[Ec. 3.9]}$$

donde:
C_i es la clase asignada por el clasificador base f_i al ejemplo x

Salida: Clasificador β

Cuando se crea una réplica, $\boldsymbol{X}_b$, del conjunto de entrenamiento inicial $\boldsymbol{X}= (x_1, x_2, ..., x_n)$, la probabilidad de que la observación i-ésima x_i (i=1,2,...,n) sea incluida m veces ($m=1,2,..,n$) en la muestra X_b viene dada por la distribución binomial B (n, $1/n$) donde $1/n$ es la probabilidad que tiene x_i de ser seleccionada en cada extracción y n es el número de muestreos con remplazamiento que se efectúan. En concreto la probabilidad de que x_i sea extraída m veces será:

$$P(m) = \binom{n}{m}\left(\frac{1}{n}\right)^{m}\left(1-\frac{1}{n}\right)^{n-m} \qquad \text{[Ec. 3.10]}$$

Cuando $1/n<0'1$, es decir, para conjuntos de más de diez observaciones, se puede aproximar la distribución binomial a través de la Poisson, P(8), donde 8=n(1/n)=1, y en consecuencia la probabilidad de que de que X_i sea extraída *m* veces es:

$$P(m) = \frac{e^{-1}}{m!} \qquad \text{[Ec. 3.11]}$$

De este modo cada observación tiene una probabilidad aproximada de *1/e* de no ser incluida (sustituyendo en la ecuación anterior m=0) en una réplica *bootstrap*.

Aunque los clasificadores base construidos siguiendo este método no siempre son mejores que el que se obtendría usando todo el conjunto de ejemplos, en [Breiman, 1996] se muestra que *Bagging* funciona bien cuando los clasificadores base se construyen usando algoritmos de aprendizaje *inestables*. Es decir, algoritmos que ante pequeñas variaciones del conjunto de datos de entrenamiento generan modelos con grandes diferencias. Ejemplos de este tipo de algoritmos son los árboles de decisión, las redes de neuronas artificiales y los algoritmos de inducción de reglas. En cambio, cuando el algoritmo de aprendizaje es *estable* (p.e. los métodos de regresión lineal o el vecino más cercano) los resultados obtenidos por *Bagging* pueden ser peores que los que se obtendrían usando un único clasificador. No obstante, en [Skurichina & Duin, 2001] se defiende que la estabilidad de los clasificadores depende del tamaño del conjunto de entrenamiento. Por tanto, y según estos autores, la utilidad de *Bagging* no está vinculada con el tipo de clasificador sino que depende de la aplicación concreta a la que se enfrente.

3.4. Uno Contra Todos - OAA

El método más simple, popular, y posiblemente más criticado, desarrollado para resolver problemas de clasificación multiclase a través de la manipulación de las salidas es el llamado **Uno Contra Todos** (*OAA – One Against All*). Dado un problema con *k* clases, se crean *k* clasificadores binarios, cada uno de ellos reconociendo una clase frente a las (*k-1*) restantes. Cuando un ejemplo es procesado, cada clasificador del sistema produce una salida. Éstas se comparan y se toma, como salida final del sistema, la clase asociada al clasificador que aportó la salida con mayor valor:

$$c(\bar{x}) = \arg\max_{i=1,\ldots,k}(y_i) \qquad \text{[Ec. 3.12]}$$

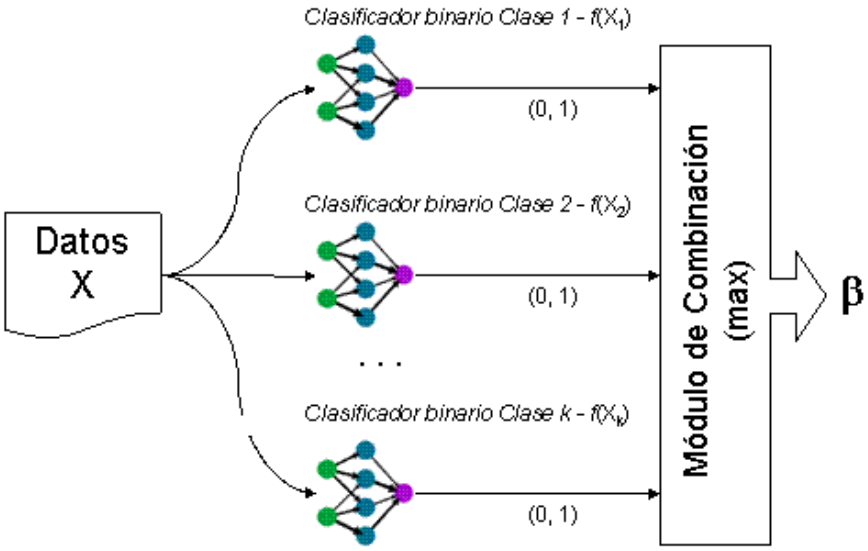
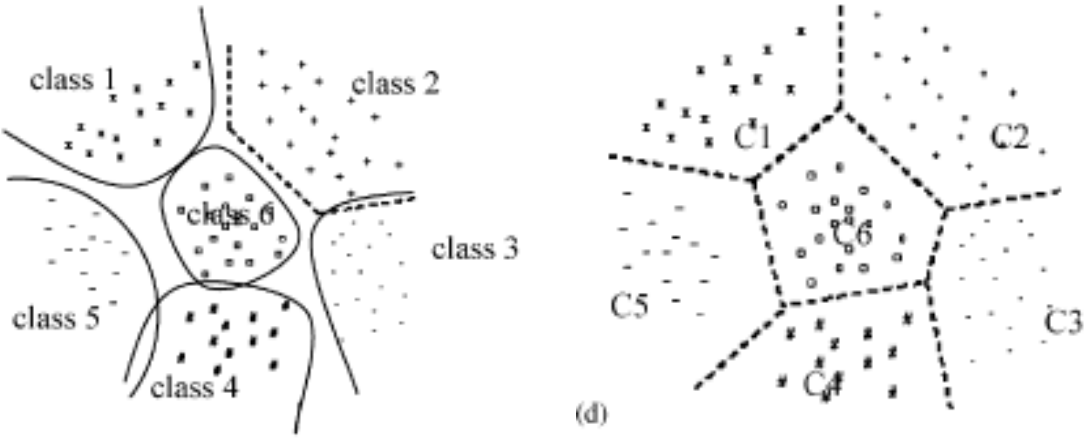


Figura 3.5. Diagrama de un sistema Uno Contra Todos (OAA) de k clases

La principal desventaja de esta arquitectura es que en ella la diversidad de los clasificadores base no siempre implica una mejora en la clasificación del conjunto [Sesmero et al., 2010]. No obstante su versatilidad, simplicidad y la calidad de sus resultados, en ocasiones comparable a la de otros métodos más sofisticados [Rifkin & Klautau, 2004], lo han convertido en uno de los métodos más usados a la hora de resolver problemas multi-clase.



Esquema de clasificación usando 6 redes binarias con arquitectura OAA

Esquema de clasificación óptimo

Figura 3.6. Comparación esquemas de clasificación OAA - optimo

3.5. Codificación de la Salida para la Corrección de Errores - ECOC

Otro método de descomposición de un problema multiclase en varias tareas de clasificación binaria es el denominado *Codificación de la Salida para la Corrección de Errores* (en adelante ECOC - *Error-Correcting Output-Coding*).

La base de este método está en la descomposición de un problema de K clases en N subproblemas binarios. En cada uno de estos subproblemas el conjunto de datos inicial, C, se divide en dos subconjuntos C_j^+ y C_j^-. Los ejemplos asociados a las clases contenidas en C_j^+ son re-etiquetados con 1 y las instancias de cualquier clase contenida en C_j^- son re-etiquetados con 0. Al repetir este proceso N veces se obtienen N conjuntos de entrenamiento distintos y en consecuencia, N clasificadores distintos, fn. Al usar un mayor número de módulos (N) que de clases (K), el sistema clasificador puede aumentar su tolerancia a fallos [Dietterich & Bakiri, 1995].

En la tabla 3.1 se muestra la matriz de descomposición o matriz ECOC para un problema de $K = 5$ clases y $N = 7$ clasificadores.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
Clase 1	0	0	0	0	0	0	0
Clase 2	0	1	1	0	0	1	1
Clase 3	0	1	1	1	1	0	0
Clase 4	1	0	1	1	0	1	0
Clase 5	1	1	0	1	0	0	1

Tabla 3.1. Ejemplo de matriz ECOC

Una vez creados los clasificadores base, ECOC clasifica una nueva instancia $\boldsymbol{x}$ a partir de la salida generada por estos clasificadores. La reconstrucción del problema (Fig. 3.7.) pasa por admitir que la instancia pertenecerá a aquella clase que, en términos de la distancia Hamming, se aproxima más a la salida generada por el conjunto. Es decir, una vez conocida la salida del sistema, se calcula la distancia Hamming (número de bits en que difieren) entre ésta y el código asociado a las distintas clases (filas de la matriz ECOC). La clase con menor distancia Hamming será seleccionada como la predicción del conjunto.

Así, si en el ejemplo anterior la salida dada por el conjunto fuera (1, 1, 0, 0, 0, 0, 1) tras calcular la distancia Hamming al código asignado a cada clase (tabla. 3.1.) se concluiría, como se muestra en la Tabla 3.2., que la instancia pertenece a la clase 5.

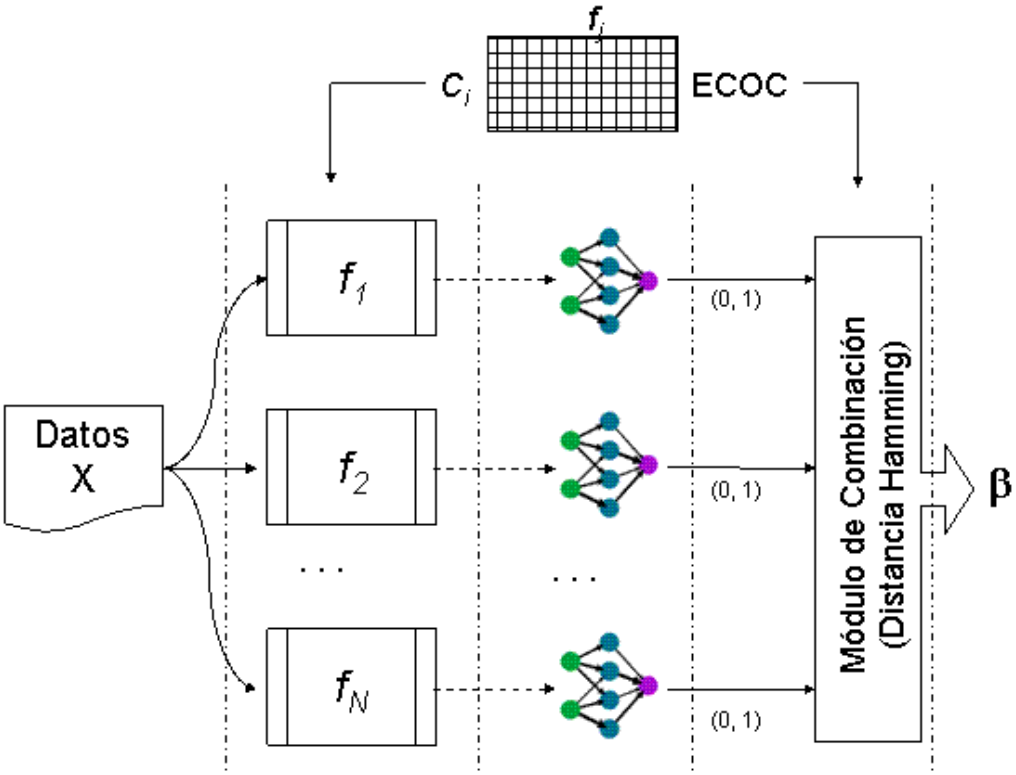


Figura 3.7. Diagrama de funcionamiento de un sistema ECOC

Salida	***1***	***1***	***0***	***0***	***0***	***0***	***1***	*D. Hamming*
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	
Clase 1	0	0	0	0	0	0	0	3
Clase 2	0	1	1	0	0	1	1	3
Clase 3	0	1	1	1	1	0	0	5
Clase 4	1	0	1	1	0	1	0	4
Clase 5	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**

Tabla 3.2. Ejemplo de cálculo de la distancia Hamming

Para aquellos casos en los que la salida de los clasificadores base es un número real comprendido en el intervalo [0,1], los autores de ECOC proponen sustituir la distancia Hamming por la distancia L^1. En este caso, la clase predicha por el sistema estará dada por:

$$C(\bar{x}) = \arg\min_{j=1,..,k} \sum_{i=1}^{N} \left| y_i - f_{i,j} \right| \qquad \text{[Ec. 3.13]}$$

donde:

y_i es la salida dada por el i-ésimo clasificador base

$f_{i,j}$ es el j-ésimo elemento de la i-ésima fila de la matriz ECOC

La principal dificultad que presenta el uso de esta técnica está en la obtención de un código ECOC capaz de subsanar los errores cometidos por los clasificadores individuales. Para ello, el código ECOC debe satisfacer dos propiedades:

Separación entre filas: Si la mínima distancia entre dos palabras del código es *d*, ECOC puede corregir hasta *(d-1)/2* errores simples (bits erróneos)

Separación entre columnas. Para garantizar una baja correlación entre los errores cometidos por los distintos clasificadores se requiere que la separación entre columnas, en términos de la distancia Hamming, sea grande. En caso contrario, es decir, si dos comunas *i* y *j* son similares o idénticas, los clasificadores *fi* y *fj* tendrán un comportamiento análogo. En ese caso cometerán los mismos errores y éstos serán difícilmente rectificables.

3.6. Selección de características

La Selección de Características (*Feature Selecttion, Attribute Selection*) es un proceso comúnmente utilizado en Aprendizaje Automático que consiste en identificar y eliminar de los datos de entrada tanta información redundante e innecesaria como sea posible. El mejor subconjunto de datos contendrá el menor número de atributos que más contribuya a la precisión del aprendizaje. Este proceso reduce la dimensionalidad de los datos permitiendo a los algoritmos de aprendizaje aumentar su eficacia y rapidez [Hall, 1999].

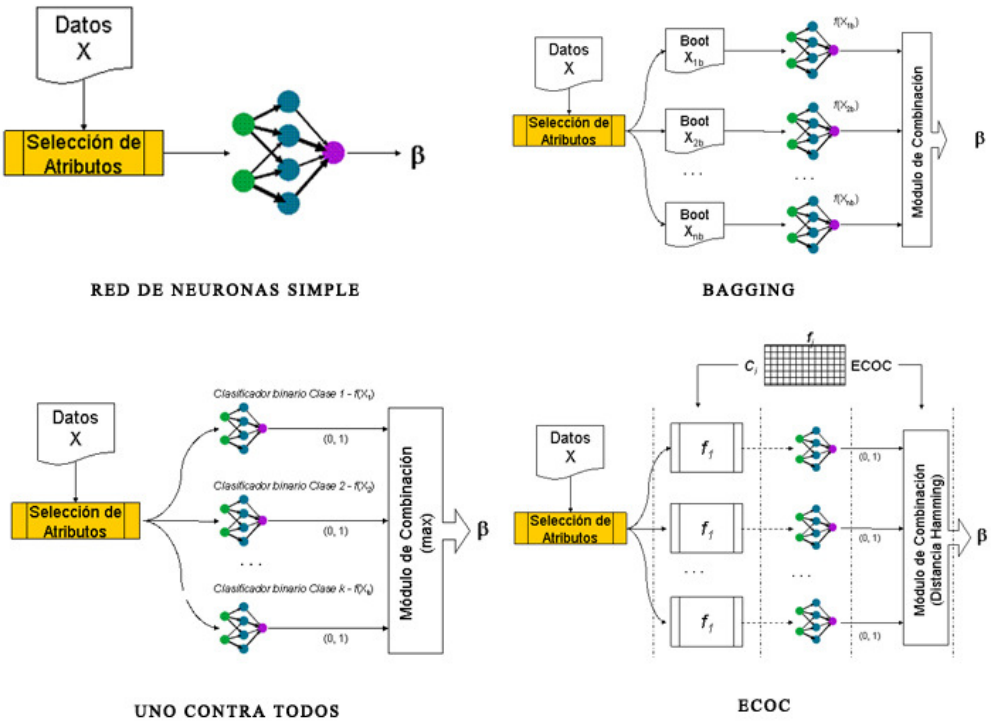


Figura 3.8. Selección de Atributos aplicada a los métodos implementados

La Selección de características en los procesos de aprendizaje automático tiene, como principales ventajas, las siguientes [Guyon & Elisseeff, 2003]:

- Al mantener un menor número de atributos y todos relevantes, facilita la visión y comprensión de los datos. Se pretende ayudar a la major comprensión de los datos mostrando qué atributos son los importantes y como están relacionados con los otros
- Reduce los requerimientos de almacenaje, reduciendo a su vez los tiempos de las fases de aprendizaje y pruebas, con la consiguiente mejora del rendimiento de las predicciones

Existen diversos métodos o algoritmos para la selección de características, siendo el más sencillo la selección aleatoria. Este es el método implementado en este proyecto.

3.7. Comparación estadística de los sistemas de clasificación.

Una cuestión importante tras la construcción de dos o más modelos de clasificación es determinar cuál de ellos resultará más fiable al aplicarlo sobre instancias de las que no se ha aprendido. Aunque dar respuesta a esta cuestión es prácticamente una utopía existen estadísticos que permiten estimar si el comportamiento de los clasificadores es equivalente o si por el contrario uno de ellos parece más fiable que los restantes.

A continuación se describen dos de estos estadísticos.

3.7.1. Test 5 x 2 cv-*F*

El método más utilizado a la hora de estimar la precisión de un clasificador es el de validación cruzada con *k* particiones (k-cv) [Stone, 1978]. Este método consiste en dividir el conjunto de datos inicial, *D*, en *k* subconjuntos disjuntos D_1, …, D_k. En cada iteración *i* (que varía de 1 a *k*), el algoritmo se entrena con el conjunto $D \backslash D_i$ y se evalúa sobre D_i. Al combinar los resultados obtenidos en las distintas iteraciones es posible estimar la precisión del sistema. Si este proceso se repite con dos modelos de clasificación distintos el estadístico *t* nos permite determinar si las diferencias observadas en los resultados son o no estadísticamente significativas.

Dietterich [Dietterich, 1998] analizó el comportamiento de la validación cruzada con k particiones combinada con el estadístico *t*. En ese trabajo propuso modificar el método estadístico utilizado y justifica que es más efectivo realizar k/2 ejecuciones del proceso de validación cruzada con 2 particiones que realizar validación cruzada con k particiones. Como solución de compromiso entre la potencia del test y el tiempo de cálculo, propone realizar 5 ejecuciones del proceso de validación cruzada con 2 particiones: 5x2cv. En cada una de las 5 iteraciones, los datos se dividen aleatoriamente en dos subconjuntos de igual cardinalidad. Después, cada uno de estos subconjuntos se

emplea para construir un clasificador y el otro para evaluarlo (Figura 3.10). Bajo estas condiciones y admitiendo que $p_i^{(j)}$ es la diferencia observada en el porcentaje de error cometido por dos clasificadores al evaluarlo sobre el subconjunto *j* de la iteración *i*, $\overline{p}_i = (p_i^{(1)} + p_i^{(2)})/2$ es la media y $s_i^2 = (p_i^{(1)} - \overline{p}_i)^2 + (p_i^{(2)} - \overline{p}_i)^2$ la varianza observada, entonces:

$$\tilde{t} = \frac{p_1^{(1)}}{\sqrt{\frac{1}{5}\sum_{i=1}^{5} s_i^2}} \qquad \text{[Ec. 3.14]}$$

sigue una distribución t con 5 grados de libertad.

En [Alpaydin, 1999] se propone reemplazar el estadístico t por una variante que no depende del orden en que se realizan los experimentos. En su trabajo, se define un nuevo estadístico que según su investigación, es más robusto que el propuesto por Dietterich. Según este autor, si se parte de los resultados derivados del proceso 5x2cv:

$$f = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}(p_i^{(j)})^2}{2\sum_{i=1}^{5} s_j^2} \qquad \text{[Ec. 3.15]}$$

sigue aproximadamente una distribución F con 10 y 5 grados de libertad. Por tanto, es posible rechazar la hipótesis nula (los dos algoritmos tienen el mismo nivel de error) con un grado de confianza de 0.95 siempre que el valor de f **sea mayor que** $\mathbf{F_{0.05}(10,5)=4.74}$**.** En tal caso, el mejor método es aquel que menos errores cometió.

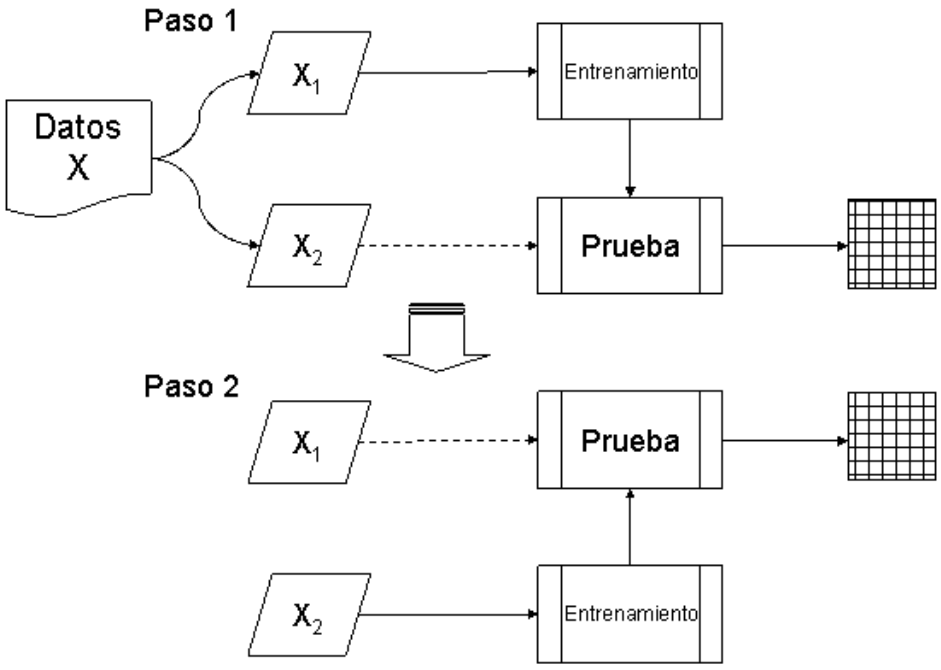


Figura 3.9. Esquema de Validación Cruzada

3.7.2. Test de McNemar

Para aplicar el test de McNemar [McNemar, 1947], el conjunto de datos se divide en dos subconjuntos: entrenamiento y test. Después, y partiendo del conjunto de datos de entrenamiento se construyen dos clasificadores f_a y f_b, que posteriormente son evaluados sobre el conjunto de test.

Así, si:

n_{00}: Número de ejemplos clasificados erróneamente por ambos sistemas f_a y f_b

n_{01}: Número de ejemplos clasificados erróneamente por f_a pero no por f_b

n_{10}: Número de ejemplos clasificados erróneamente por f_n pero no por f_a

n_{11}: Número de ejemplos clasificados correctamente por ambos sistemas.

y se acepta que bajo la hipótesis nula (los clasificadores son equivalentes) los dos clasificadores deben tener la misma tasa de error, se puede afirmar que cuando

$$X^2 = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}} \qquad \text{[Ec. 3.16]}$$

tiene un valor inferior a $\chi^2_{1,0.95}$ = 3.841459, la probabilidad de que los sistemas sean equivalentes es superior a 0.95.

Los resultados obtenidos de este test tienen menos fiabilidad que los obtenidos a través del 5x2cv F_Test, pero es una alternativa a éste cuando la elevada dimensionalidad del problema hace que no sea viable la ejecución del proceso de validación cruzada [Dietterich, 1998].

Capítulo 4

Evaluación Experimental

En este capítulo se detalla el trabajo experimental realizado durante este proyecto con las distintas opciones disponibles en la aplicación desarrollada y se exponen los resultados obtenidos.

En primer lugar se informa sobre las pruebas realizadas para consolidar el desarrollo de la aplicación. A continuación se presenta la interfaz de la aplicación desarrollada. Seguidamente se define el dominio de estudio, se detallan las particularidades de los modelos implementados al aplicarlos sobre dicho dominio y se presentan y analizan los resultados experimentales obtenidos.

4.1. Pruebas de sistema y consolidación

La primera etapa del desarrollo consistió en identificar, definir e implementar las clases y métodos necesarios para componer un Perceptrón Multicapa en el que el aprendizaje se materializa haciendo uso del algoritmo de retro-propagación (*Back-Propagation*).

Dado que la comprobación matemática de este algoritmo es costosa y complicada y realizarla de forma manual es prácticamente imposible, en la etapa de validación, se ha tomado como referencia el proyecto de la Universidad de Stutgart: Simulador de Redes de Neuronas de Stutgart (*Stutgart Neural Network Simulator* - *www.ra.cs.uni-tuebingen.de/SNNS*), que ofrece una implementación de éste algoritmo. Para facilitar la comparación entre ambos proyectos, los ficheros de datos usados en este trabajo se ajustan a la estructura y formato utilizados por el *SNNS*. Generado el fichero de datos, el proceso de prueba se puede esquematizar en los siguientes puntos:

- Se genera una red inicial con valores aleatorios desde nuestra aplicación y se guarda según el formato definido por el proyecto *SNNS*
- Se toman determinados valores para los parámetros variables de los procesos: coeficiente de aprendizaje, número de ciclos de aprendizaje, etc.
- Con estos valores, y partiendo de la red inicial creada en el primer punto, se genera una Red entrenada por medio de nuestra aplicación y de la misma forma una Red entrenada por el proyecto *SNNS*. Se guardan las redes resultantes en el formato adecuado.
- Se comprueba que los archivos guardados son equivalentes. En la figura 4.1 se observa como dos redes entrenadas a partir del mismo fichero de Red Inicial por medio de la aplicación desarrollada en este proyecto y la aplicación SNNS obtienen un gráfico de evolución del error similar.

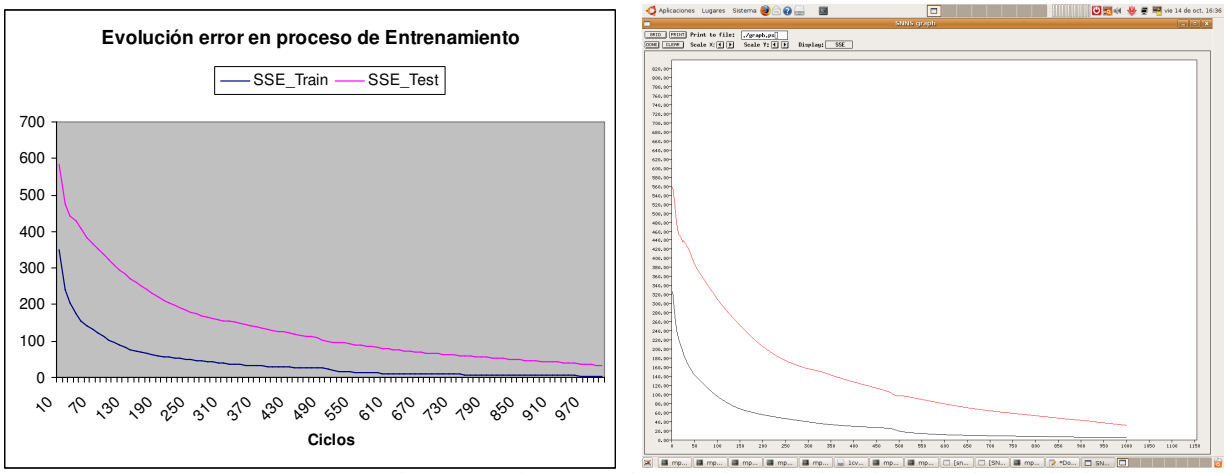


Figura 4.1. Comparación gráficas de SSE en el proceso de aprendizaje en la aplicación desarrollada y por medio de SNNS

Para facilitar el proceso de comparación y hacerlo exhaustivo, en una primera fase se utilizó un conjunto de datos simple: 4 atributos de entrada y una única salida y el número de ciclos de aprendizaje se fijó en 10. Tras lograr una coincidencia entre los datos ofrecidos por nuestra aplicación y los dados por el SNNS, se elevaron las dimensiones del dominio y la aplicación se sometió a un nuevo proceso de validación que terminó cuando los valores resultantes del entrenamiento eran los deseados. Validada la fase de entrenamiento se procedió a verificar el proceso de test hasta comprobar que tanto el error cuadrático medio obtenido como el número de aciertos/errores coincidían con los dados por la aplicación de referencia (Figuras 4.1 y 4.2) .

Una vez consolidada la base de nuestro sistema, se desarrollaron los procesos que implementan los métodos de construcción de Conjuntos de Clasificadores, comprobando que los pasos de cada uno ofrecen los resultados deseados. La mayor parte de los errores cometidos en esta etapa fueron detectados al analizar la matriz de confusión obtenida tras la aplicación cada método.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	**49**	0	1	0	0	0	0	0	0
C2	0	**50**	0	0	0	0	0	0	0
C3	0	0	**47**	1	0	1	1	0	0
C4	1	0	0	**48**	0	0	1	0	0
C5	0	0	0	0	**23**	4	22	0	1
C6	1	1	3	2	2	**32**	6	3	0
C7	1	0	4	1	1	0	**38**	4	1
C8	0	0	0	0	2	0	1	**47**	0
C9	0	0	4	4	1	0	1	0	**40**

Tabla 4.1. Ejemplo de Matriz de Confusión mostrando resultados de un proceso de la aplicación desarrollada (Red de Neuronas Simple, 450 ejemplos clasificados)

CONFUSION MATRIX (rows: teaching input, colums: classification)

class	0	1	2	3	4	5	6	7	8	unknown
0	49	0	1	0	0	0	0	0	0	0
1	0	50	0	0	0	0	0	0	0	0
2	0	0	47	1	0	1	1	0	0	0
3	1	0	0	48	0	0	1	0	0	0
4	0	0	0	0	23	4	22	0	1	0
5	1	1	3	2	2	32	6	3	0	0
6	1	0	4	1	1	0	38	4	1	0
7	0	0	0	0	2	0	1	47	0	0
8	0	0	4	4	1	0	1	0	40	0
noclass	0	0	0	0	0	0	0	0	0	0

Figura 4.2. Matriz de Confusión generada por SNNS sobre el mismo ejemplo de la Figura 4.2.

Por último, se implementaron los algoritmos requeridos para la comparación estadística de los métodos implementados: F_Test y McNemar. Al ser algoritmos sencillos que trabajarán con pocos datos, su funcionamiento se validó comparando los valores de entrada y los resultados obtenidos con los dados por los siguientes recursos.

- Tomando como referencia una hoja de cálculo creada a tal efecto con las fórmulas correspondientes
- Con aplicaciones online (*QuickCalcs*)
  - McNemar: http://www.graphpad.com/quickcalcs/McNemar1.cfm
  - F_test: http://www.graphpad.com/quickcalcs/AIC1.cfm

4.2. Interfaz de la aplicación

Como se puede ver en la Figura 4.3., la aplicación consta de una única pantalla muy sencilla para facilitar al usuario la ejecución de las tareas. En la parte superior, se introduce la ruta de los ficheros de entrenamiento y validación, así como la carpeta de trabajo donde la aplicación creará los ficheros resultantes del proceso de entrenamiento y test.

En la parte izquierda se encuentra la parte que se refiere a la selección de características, pudiendo activar esta opción y seleccionar el subconjunto de características aleatoriamente o cargándolo mediante un fichero.

En el centro de la pantalla se sitúan los distintos sistemas de conjuntos de clasificadores que podemos seleccionar para ejecutar un proceso o, como última opción, se puede seleccionar la comparación de los anteriores sistemas empleando los algoritmos de *F_Test 5x2cv* o *McNemar*. Justo a la derecha de estas opciones, se encuentran las casillas que cuantifican los parámetros necesarios para configurar las redes de neuronas, es decir, los valores que necesitará el sistema para procesar la información.
Las variables disponibles son:

- Número de neuronas de la capa oculta
- Coeficiente de aprendizaje: Parámetro que determina la velocidad de convergencia del algoritmo de retro-propagación (Sección 3.2.).

- Número de ciclos que realizará el algoritmo de aprendizaje en esta fase.
- Boots (solo para el método *Bagging*), número de subconjuntos en que será dividido el fichero para crear la arquitectura *Bagging*.
- SSE_End es un valor de referencia con el que se pretende evitar el sobre-entrenamiento en la fase de aprendizaje.

Por último, en la parte inferior de la interfaz se encuentran las tablas de resultados, donde se van añadiendo en nuevas pestañas los resultados obtenidos al ejecutar un proceso de clasificación.

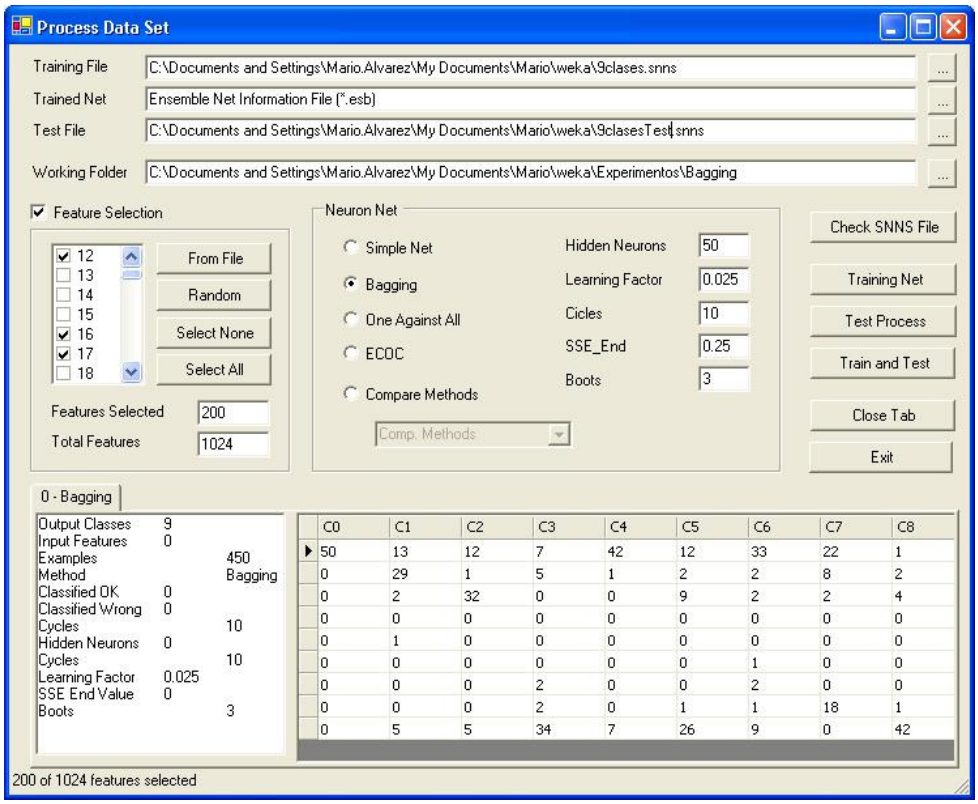


Figura 4.3. Interfaz de la aplicación

A continuación se describe el formato de los distintos ficheros manejados por la aplicación.

4.2.1. Base de datos de ejemplos (*.snns)

Como se citó en este mismo capítulo, y con el objetivo de tener una base de referencia con la que validar el funcionamiento de la aplicación, los ficheros de datos de entrenamiento y prueba están basados en el proyecto SNNS de la Universidad de Stutgart.

El formato de estos ficheros es el siguiente:

SNNS pattern definition file v3.2
generated at DDD-MMM-YYY HH:MM:SS YYYY
<Salto de línea>
<Salto de línea>
<Salto de línea>
No. of patterns : <número de patrones o ejemplos>
No. of input units : <número de atributos (features) por patrón>
No. of output units : <número de clases o salidas>
<Salto de línea>
<Salto de línea>
<comentario entrada> p.ej.: #i S60/AG00091.ppm inputs 1024 clase 5/9 AG00091.ppm
<patron1.entrada1>< patron1.entrada2> … < patron1.entradaM>
<comentario salida> p.ej: #o S60/AG00091.ppm inputs 1024 clase 5/9

<patron1.salida1><patron1.salida2> … <patron1.salidaS>
<Salto de línea>
<comentario entrada> p.ej.: #i S60/AG00091.ppm inputs 1024 clase 5/9 AG00091.ppm
<patron2.entrada1>< patron2.entrada2> … < patron2.entradaM>
<comentario salida> p.ej: #o S60/AG00091.ppm inputs 1024 clase 5/9
<patron2.salida1><patron2.salida2> … <patron2.salidaS>
…
<Salto de línea>
<comentario entrada> p.ej.: #i S60/AG00091.ppm inputs 1024 clase 5/9 AG00091.ppm
<patronNentrada1>< patronN.entrada2> … < patronN.entradaM>
<comentario salida> p.ej: #o S60/AG00091.ppm inputs 1024 clase 5/9
<patronN.salida1><patronN.salida2> … <patronN.salidaS>
<fin de fichero>

Donde *M* es el número de atributos de los ejemplos indicado en *<número de atributos>*, *S* es el número de clases del sistema indicado en *<número de clases o salidas>* y contendrá un valor de 1.0 para la clase a la que corresponde el ejemplo y 0.0 el resto de valores, y *N* es el número de ejemplos que contiene el fichero indicado en *<número de patrones o ejemplos>*.

4.2.2. Selección de características (*.txt)

El fichero para cargar una selección de atributos predeterminada es un fichero de texto de tantas líneas como atributos tengan las entradas del dominio. Cada línea tiene un valor de 0 si el atributo no es seleccionado, y 1 si el atributo es seleccionado.

4.2.3. Fichero de código ECOC (*.txt)

El fichero de códigos ECOC se carga en los procesos de la arquitectura ECOC, y se trata de un fichero de texto donde la primera línea indica el número de clases de la arquitectura, y a continuación hay tantas líneas como número de clases con tantos dígitos (0 ó 1) como número de funciones contiene el sistema, separados por un espacio. A continuación se muestra el esquema del formato del fichero de códigos ECOC:

<número de clases N>
<$C_1.f_1$><$C_1.f_2$> … <$C_1.f_F$>
<$C_2.f_1$><$C_2.f_2$> … <$C_2.f_F$>
…
<$C_N.f_1$><$C_N.f_2$> … <$C_N.f_F$>
<fin de fichero>

donde *F* es el número de funciones de la arquitectura *ECOC*

4.2.4. Fichero de Red de Neuronas, Inicial o Entrenada (*.net)

Estos ficheros almacenan las características arquitectónicas (número de neuronas en las distintas capas) y topológicas (activación, peso de las conexiones y coeficiente de aprendizaje) de la red de neuronas inicial (los pesos tienen un valor asignado de forma aleatoria) y entrenada (la resultante del proceso de aprendizaje y, por consiguiente, la usada en el proceso de evaluación). El formato de estos ficheros es el que se muestra a continuación:

Entradas: <número de entradas>
Neuronas Ocultas: <número de neuronas ocultas>
Salidas: <número de salidas>
Alfa: <coeficiente de aprendizaje>
Beta: <no utilizado>
Activación[<capa oculta.neurona1>] <valor activación>
Bias[<capa oculta.neurona1>] <valor Bias>
Peso[<capa oculta.neurona1>] [capa entrada.neurona1]<peso>
…
Peso[<capa oculta.neurona1>] [capa entrada.neuronaE]<peso>
…
Activación[<capa oculta.neuronaO>] <valor activación>
Bias[<capa oculta.neuronaO>] <valor Bias>
Peso[<capa oculta.neuronaO>] [capa entrada.neuronaO]<peso>
…
Peso[<capa oculta.neuronaO>] [capa entrada.neuronaO]<peso>
Activación[<capa salida.neurona1>] <valor activación>
Bias[<capa salida.neurona1>] <valor Bias>
Peso[<capa salida.neurona1>] [<capa salida.neurona1>]<peso>
…
Peso[<capa salida.neuronaO>] [capa salida.neuronaO]<peso>
...
<fin de fichero>

4.3.5. Fichero de detalles de arquitectura de Redes de Neuronas (*.esb)

El fichero de detalles de arquitectura de Redes de Neuronas se genera al mismo tiempo que los ficheros de Redes vistos en el punto anterior, y es un fichero de texto que contiene información acerca del conjunto de clasificadores: arquitectura utilizada, número de clasificadores o Redes de Neuronas, y la localización de los ficheros donde se guarda la información de estas Redes. Este fichero permite al proceso de prueba cargar las redes necesarias generadas durante el aprendizaje automático de la arquitectura, y su formato es el siguiente:

<arquitectura> - ECOC, Bagging, OAA, etc.
<número de Redes>
<path Fichero1.net>
<path Fichero2.net>
...
<path FicheroN.net>
<fin de fichero>
donde *N* es el número de Redes de Neuronas de la arquitectura indicado en *<número de Redes>*

4.3.6. Fichero Matriz de Confusión (*.txt)

Cada proceso de prueba de un clasificador obtiene unos resultados que son evaluados a través de una Matriz de Confusión, la cual se almacena en este fichero. Es un fichero de texto con información de la arquitectura evaluada y los valores de la Matriz de Confusión, con el siguiente formato:

Results File for <arquitectura> method generated at DDD-MMM-YYY HH:MM:SS YYYY
Nº of Examples: <número de ejemplos> - Inputs: <número de atributos> - Classes: <número de clases>
{<otras propiedades>}
|| N_{00} || N_{01} || ... || N_{0C}||
...
...
|| N_{C0} || N_{C1} || ... || N_{CC}||

donde *C* es el número de clases del dominio, y N_{ij} es el número de veces que el conjunto de clasificadores ha determinado que un ejemplo perteneciente a la clase i, pertenece a la clase j. Por lo tanto, las celdas donde i=j suponen los aciertos de la prueba, mientas que las celdas donde i≠j suman los errores de la misma.

4.3. Descripción del dominio

El dominio empleado para comprobar la viabilidad de este proyecto ha sido el de reconocimiento de señales de tráfico. Este dominio, contiene 900 señales de prohibición distribuidas equitativamente en 9 clases (Tabla 4.2). Inicialmente, cada señal está representada por una imagen en formato PGM (*Portable Gray Map*) de 32x32 pixeles, lo que permite representar cada ejemplo con un vector de 1024 atributos y las clases con un vector de 9 componentes. Por simplicidad, los valores de los atributos han sido normalizados y el fichero de datos correspondiente se ha ajustado al formato SNNS.

Clase	Señal de tráfico	
C1	Prohibido peatones	
C2	Prohibido girar a la izquierda	
C3	Prohibido aparcar	
C4	Prohibido adelantar	
C5	Prohibido circular a más de 60 km/h	
C6	Prohibido circular a más de 50 km/h	
C7	Prohibido circular a más de 40 km/h	
C8	Prohibido circular a más de 20 km/h	
C9	Prohibido circular a más de 100 km/h	

Tabla 4.2. Señales de tráfico incluidas en el dominio

4.4. Características de los modelos implementados

En este apartado se detallan las características específicas de los modelos implementados cuando éstos se aplican al reconocimiento de señales de tráfico.

En lo que respecta a las Redes de Neuronas Artificiales, el sistema está restringido a una sola capa oculta con un número de neuronas variable, cuyo valor en nuestros experimentos se ha fijado en **50**. El valor del coeficiente de aprendizaje utilizado es **0,025** y el número de ciclos se establece en **1000**.

La arquitectura de *Bagging* se construye con **15** clasificadores base, por lo que el fichero de entrada queda dividido en 15 subconjuntos o *boots*.

Para *ECOC*, el número de clasificadores base se ha fijado en 15 y la matriz ECOC usada es la que se muestra en la Tabla 4.3.

	f1	*f2*	*f3*	*f4*	*f5*	*f6*	*f7*	*f8*	*f9*	*f10*	*f11*	*f12*	*f13*	*f14*	*f15*
C1	0	1	0	0	1	0	1	0	1	1	0	0	1	0	1
C2	0	0	1	1	1	1	0	1	0	1	1	1	1	0	0
C3	1	0	0	1	0	0	0	1	1	1	1	0	1	1	0
C4	0	0	1	1	0	1	1	1	0	0	0	0	1	1	1
C5	1	1	1	0	1	0	1	1	0	0	1	0	0	1	0
C6	0	1	0	0	1	1	0	1	1	1	0	0	0	1	1
C7	1	0	1	1	1	0	1	0	0	1	0	0	0	1	1
C8	0	0	0	1	1	1	1	0	1	0	1	0	0	1	0
C9	1	1	0	1	0	0	1	1	0	1	0	1	0	0	0

Tabla 4.3. Matriz ECOC empleada en la construcción de la arquitectura

Para la selección de atributos se ha optado por fijar el número de características seleccionadas en **200.**

Para comparar los resultados de los métodos propuestos se utilizan los tests de McNemar y F_Test, este último empleando la técnica de validación cruzada (cross validation) de 5x2cv, es decir, dos pares de ficheros balanceados (cada uno con el mismo número de ejemplos de cada clase), 5 repeticiones. Las pruebas para obtener los valores para el test de McNemar se realizan con los ficheros obtenidos a partir del fichero de entrada, con 2/3 de los ejemplos para entrenar el Conjunto de Clasificadores y 1/3 para la fase de tests.

4.5. Valores de Precisión obtenidos

Fijados los parámetros de los distintos sistemas a los valores indicados en el apartado anterior, los valores de precisión obtenidos para cada modelo son los resumidos en la figura 4.4. Todos los valores mostrados en esta sección se han obtenido tras dividir el conjunto de datos inicial en dos subconjuntos: Entrenamiento y Test. Para el fichero de Entrenamiento se han seleccionado aleatoriamente 2/3 del conjunto de datos inicial reservando el resto de ejemplos (1/3) para el fichero de Test.

En los siguientes sub-apartados se muestra la matriz de confusión obtenida al evaluar los distintos modelos sobre el conjunto de Test. En cada matriz, las filas representan la salida esperada para los distintos ejemplos y las columnas la clase asignada por el clasificador. Por tanto, los valores de la diagonal principal indican, para cada clase, el número de ejemplos correctamente clasificados por cada sistema.

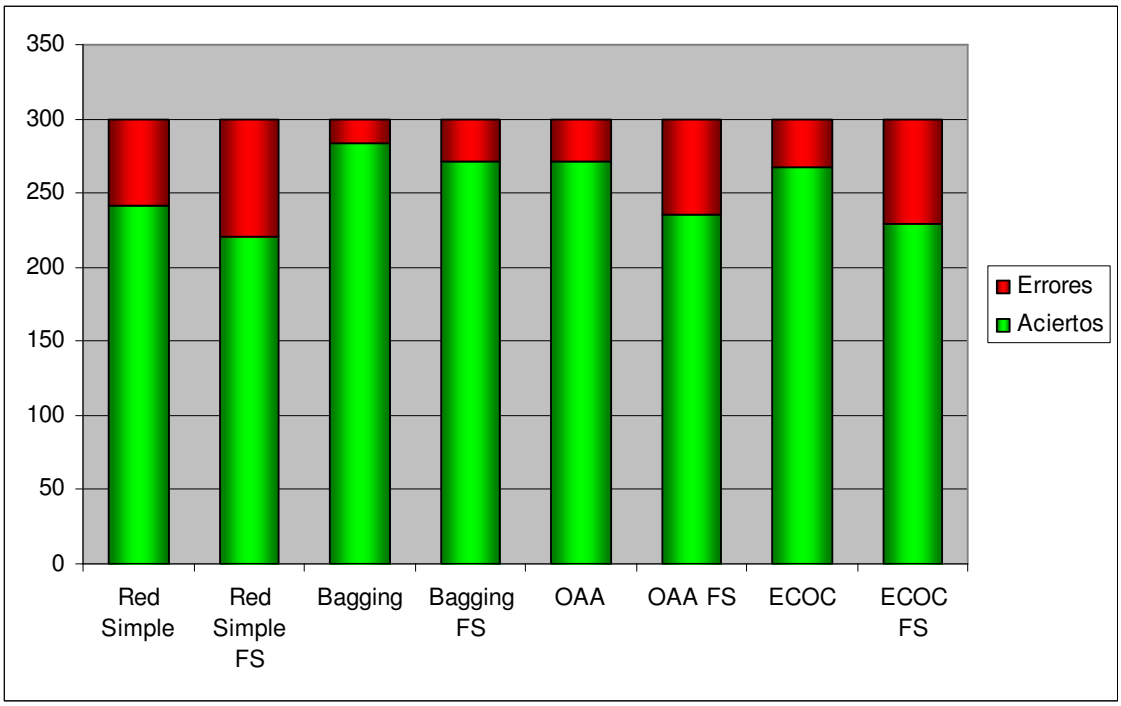


Figura 4.4. Resultados de precisión de los métodos para evaluación de 300 ejemplos

4.5.1. Red Simple

Neuronas ocultas **50**
Coeficiente de aprendizaje **0,025**
Ciclos de aprendizaje **1000**.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	33	0	0	0	0	0	0	0	0
C2	0	33	0	0	0	0	0	0	0
C3	0	0	32	0	0	0	2	0	0
C4	0	0	0	30	0	2	0	0	1
C5	0	0	0	1	2	4	25	0	1
C6	0	0	1	0	0	27	3	2	1
C7	0	0	1	0	0	0	29	3	0
C8	0	0	0	0	0	1	5	27	0
C9	0	0	0	3	0	2	0	0	29

Tabla 4.4. Matriz de Confusión obtenida de la evaluación de una Red de Neuronas Simple

Número de aciertos: 242 (80.67 %)
Número de errores: 58 (19.33 %)

Con Selección de 200 Atributos

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	32	0	0	0	0	0	1	0	0
C2	0	33	0	0	0	0	0	0	0
C3	0	0	29	1	0	3	0	1	0
C4	0	0	0	29	0	1	0	2	1
C5	0	0	0	0	6	3	21	3	0
C6	0	0	0	1	1	18	12	2	0
C7	0	0	1	0	0	0	12	20	0
C8	0	0	0	0	0	1	1	31	0
C9	0	1	0	2	0	1	0	0	30

Tabla 4.5. Matriz de Confusión obtenida de la evaluación de una Red de Neuronas Simple con Selección de Atributos

Número de aciertos: 220 (73.33 %)
Número de errores: 80 (26.67 %)

4.5.2. Bagging

Neuronas ocultas **50**
Coeficiente de aprendizaje **0,025**
Ciclos de aprendizaje **1000**.
Número de clasificadores base: **15**

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	33	0	0	0	0	0	0	0	0
C2	0	33	0	0	0	0	0	0	0
C3	0	0	34	0	0	0	0	0	0
C4	0	0	0	32	0	1	0	0	0
C5	0	0	0	0	32	1	0	0	0
C6	0	0	2	0	1	27	1	0	3
C7	0	0	0	0	1	0	30	1	1
C8	0	0	0	0	0	0	2	31	0
C9	0	0	0	2	0	0	1	0	31

Tabla 4.6. Matriz de Confusión obtenida de la evaluación de Bagging

Número de aciertos: 283 (94.33 %)
Número de errores: 17 (5.67 %)

Con Selección de 200 Atributos

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	31	0	1	0	0	1	0	0	0
C2	0	33	0	0	0	0	0	0	0
C3	0	0	32	1	0	1	0	0	0
C4	0	0	0	31	0	1	0	1	0
C5	0	0	0	0	33	0	0	0	0
C6	0	0	0	1	6	22	4	0	1
C7	0	0	1	0	0	0	32	0	0
C8	0	0	0	1	1	0	4	27	0
C9	0	0	0	1	0	1	0	2	30

Tabla 4.7. Matriz de Confusión obtenida de la evaluación de Bagging con Selección de Atributos

Número de aciertos: 271 (90.33 %)
Número de errores: 29 (9.67 %)

4.5.3. Uno Contra Todos - OAA

Neuronas ocultas **50**
Coeficiente de aprendizaje **0,025**
Ciclos de aprendizaje **1000**.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	33	0	0	0	0	0	0	0	0
C2	0	33	0	0	0	0	0	0	0
C3	0	0	30	0	0	1	1	1	1
C4	0	0	0	33	0	0	0	0	0
C5	0	0	0	0	25	2	5	0	1
C6	0	1	1	1	0	26	1	1	3
C7	0	0	0	0	0	0	28	4	1
C8	0	0	0	0	0	0	1	31	1
C9	0	0	0	2	0	0	0	0	32

Tabla 4.8. Matriz de Confusión obtenida de la evaluación de Uno Contra Todos OAA

Número de aciertos: 271 (90.33 %)
Número de errores: 29 (9.67 %)

Con Selección de 200 Atributos

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	31	0	0	0	1	0	0	1	0
C2	0	33	0	0	0	0	0	0	0
C3	0	0	31	1	0	1	0	1	0
C4	0	0	0	29	0	1	0	1	2
C5	0	0	0	0	19	0	5	9	0
C6	0	0	0	1	1	19	9	2	2
C7	0	0	1	0	0	0	14	18	0
C8	0	0	0	0	0	1	3	29	0
C9	0	0	0	1	0	1	1	1	30

Tabla 4.5. Matriz de Confusión obtenida de la evaluación de OAA con Selección de Atributos

Número de aciertos: 235 (78.33 %)
Número de errores: 65 (21.67 %)

4.5.4. ECOC

Neuronas ocultas **50**
Coeficiente de aprendizaje **0,025**
Ciclos de aprendizaje **1000**.
Número de clasificadores base (columnas de la matriz ECOC): **15**

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	33	0	0	0	0	0	0	0	0
C2	0	33	0	0	0	0	0	0	0
C3	0	0	31	0	0	0	0	2	1
C4	0	0	0	32	0	0	0	1	0
C5	1	0	0	0	21	0	2	9	0
C6	0	0	2	1	0	27	1	3	0
C7	0	0	0	0	0	0	26	6	1
C8	0	0	0	0	0	0	0	33	0
C9	0	0	0	1	0	0	1	0	32

Tabla 4.10. Matriz de Confusión obtenida de la evaluación de ECOC

Número de aciertos: 268 (89.33 %)
Número de errores: 32 (10.67 %)

Con Selección de 200 Atributos

	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	31	0	0	0	0	0	0	2	0
C2	0	33	0	0	0	0	0	0	0
C3	0	0	30	1	0	0	1	2	0
C4	0	0	0	30	0	0	1	1	1
C5	0	0	0	0	15	0	5	13	0
C6	1	1	0	3	1	11	9	6	2
C7	0	0	0	0	0	0	17	15	1
C8	0	0	0	0	0	1	1	31	0
C9	0	0	0	0	1	0	1	1	31

Tabla 4.11. Matriz de Confusión obtenida de la evaluación de ECOC con Selección de Atributos

Número de aciertos: 229 (76.33 %)
Número de errores: 71 (23.67 %)

4.6. Resultados de los tests estadísticos de comparación

Para comparar los resultados obtenidos por los distintos métodos de clasificación implementados se ha optado por usar dos de los test estadísticos más valorados en la Bibliografía: El test F y el Test de McNemar (ver sección 3.7)

4.6.1. F_Test 5x2cv

Como se señaló en el capítulo 3, para aplicar este test se requiere realizar 5 iteraciones del proceso de validación cruzada con 2 particiones y determinar el número de errores cometido por cada método en cada iteración. Los parámetros usados en este proceso coinciden con los indicados previamente en este mismo capítulo:

50 neuronas ocultas
1000 ciclos de entrenamiento
Coeficiente de aprendizaje igual a 0,025
Número de clasificadores base para *Bagging* igual a 15.
ECOC con 15 clasificadores base con la codificación recogida en la Tabla 4.3.

Tras crear, entrenar y testear las redes en las distintas iteraciones del proceso de validación cruzada, los resultados obtenidos son los recogidos en las Tablas 4.12. y 4.13. En estas tablas, se presentan el número de aciertos y errores obtenidos por cada modelo en cada iteración

Iteración	Red Simple	Bagging	OAA	ECOC
0A	367	406	376	395
0B	374	411	389	392
1A	395	407	384	393
1B	363	404	383	405
2A	381	406	401	399
2B	344	416	369	381
12A	362	409	397	407
3B	372	410	398	398
4A	373	403	394	396
4B	368	415	391	386

Tabla 4.12. Número de aciertos de la prueba de cada fichero por método

Iteración	Red Simple	Bagging	OAA	ECOC
0A	83	44	74	55
0B	76	39	61	58
1A	55	43	66	57
1B	87	46	67	45
2A	69	44	49	51
2B	106	34	81	69
3A	88	41	53	43
3B	78	40	52	52
4A	77	47	56	54
4B	82	35	59	64

Tabla 4.13. Número de errores de la prueba de cada fichero por método

Por tanto el total de aciertos y fallos por método es el que se muestra a continuación en la Tabla 4.14.

	Red Simple	Bagging	OAA	ECOC
Total Aciertos	3699	4087	3882	3952
% Aciertos	82,20%	90,82%	86,27%	87,82%
Total Fallos	801	413	618	548
% Fallo	17,80%	9,18%	13,73%	12,18%

Tabla 4.14. Aciertos y errores totales por método en el proceso de Validación Cruzada

A partir de estos valores, concretamente con el número de errores recogido en la tabla 4.13, y haciendo uso de la ecuación 3.15, se obtienen los valores para F-test mostrados en la Tabla 4.15.

	Red Simple	Bagging	OAA	ECOC
Red Simple		5.048	4.266	2.918
Bagging	5.048		2.689	1.805
OAA	4.266	2.689		1.574
ECOC	2.918	1.805	1.574	

Tabla 4.15. Resultados de F_Test de comparación de los métodos

A partir de estos valores se puede deducir que, en el dominio de estudio, los métodos de clasificación analizados son estadísticamente equivalentes, ya que el valor del test F es menor que *4.74*, excepto al comparar la *Red Simple* con *Bagging*. En este último caso, el valor obtenido (F = 5.048) indica que uno de los dos métodos es estadísticamente mejor que el otro. En concreto, se puede afirmar que *Bagging* es más preciso, ya que solo cometió un 9.18 % de error frente al 17.80 % de la *Red Simple*.

En la siguiente figura se muestra la aplicación desarrollada una vez terminado el proceso de validación cruzada y el cálculo de los valores de *F Test*

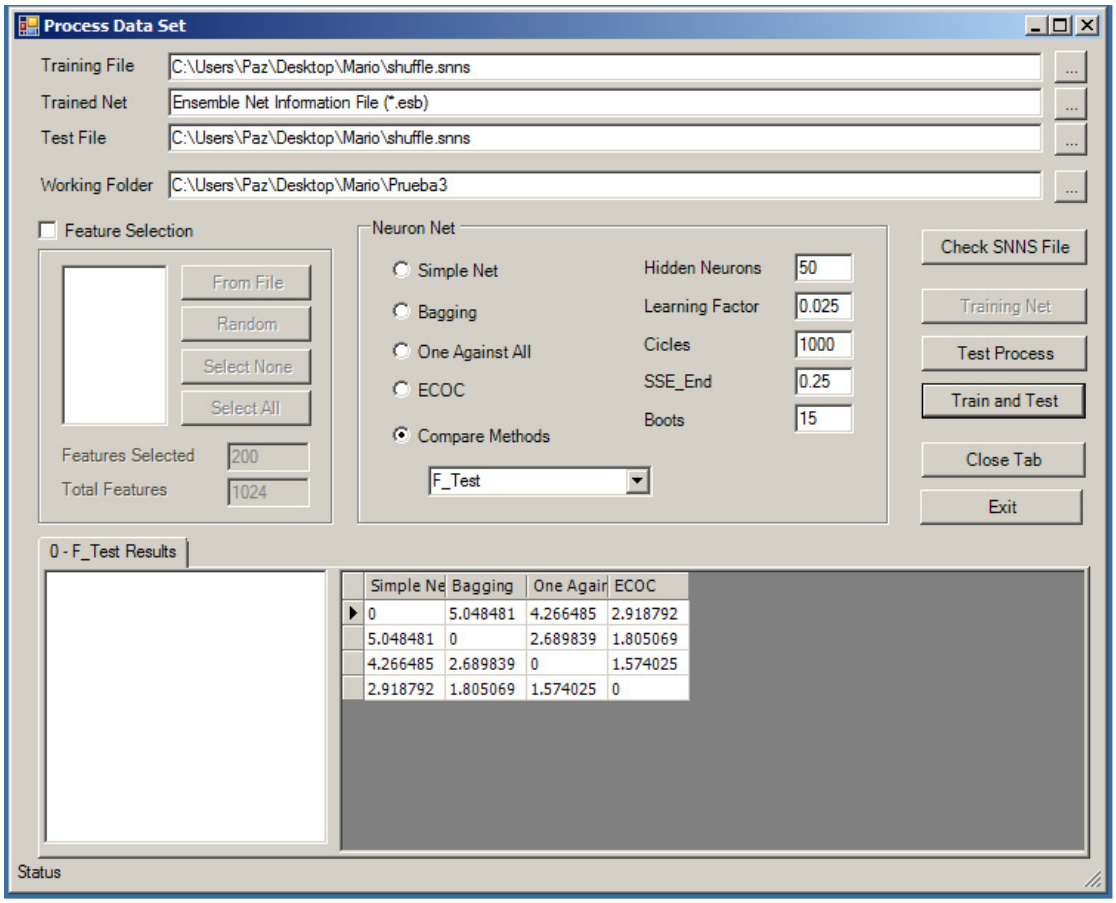


Figura 4.5. Resultados del cálculo de F Test

4.6.2. Test estadístico de McNemar

Tomando como partida el fichero de datos de 900 ejemplos y con los mismos valores que se han venido indicando para los parámetros del sistema, se construyen los distintos modelos de clasificación implementados en la aplicación. En este caso, el procedimiento seguido para obtener los ficheros de Entrenamiento y Test coincide con el detallado en el epígrafe 4.5. Es decir, el fichero de entrenamiento se obtiene seleccionando 2/3 del conjunto de ejemplos inicial y el resto de ejemplos se reservan para el fichero de test.

Una vez creados los distintos modelos, la aplicación del test de McNemar para comparar los modelos A y B, requiere la obtención de la siguiente matriz de contingencia:

Número de ejemplos mal clasificados por ambos modelos A y B	Número de ejemplos bien clasificados por B pero no por A
Número de ejemplos bien clasificados por A pero no por B	Número de ejemplos bien clasificados por ambos modelos A y B

Tabla 4.16. Descripción de la tabla empleada para calcular el valor estadístico de McNemar

A continuación se indican los modelos comparados, se muestra la matriz de contingencia obtenida en cada caso y se analiza el valor resultante de aplicar la ecuación 3.16.

a) Red Simple / Bagging

234	8
49	9

Tabla 4.17. Resultados para el cálculo de McNemar de una Red Simple contra Bagging

De estos valores se deriva que χ^2=28.070 por lo que se deduce que estos modelos no son equivalentes, siendo *Bagging* el conjunto de clasificadores mejor evaluado en la comparación, ya que cometió menor número de errores.

b) Red Simple / Uno Contra Todos:

233	9
38	20

Tabla 4.18. Resultados para el cálculo de McNemar de una Red Simple contra Uno Contra Todos OAA

El valor obtenido al aplicar el test es de 16.680. Dado que el método OAA comete un menor número de errores y el valor de χ^2 es superior a 3.841 se puede concluir que OAA es mejor que la Red Simple.

c) Red Simple / ECOC:

231	11
37	21

Tabla 4.19. Resultados para el cálculo de McNemar de una Red Simple contra ECOC

El valor obtenido al aplicar el test es de 13.020, con lo que de nuevo la Red Simple es la peor evaluada y el método ECOC es mejor al cometer menor número de errores.

d) Bagging / Uno Contra Todos:

265	18
6	11

Tabla 4.20. Resultados para el cálculo de McNemar de Bagging contra OAA

El valor obtenido al aplicar el test es de 5.041, por lo que, de nuevo, se puede concluir que entre ambos sistemas existen diferencias significativas. Además, y puesto que Bagging es más preciso, se puede concluir que este método es el mejor de los dos.

e) Bagging / ECOC:

260	23
8	9

Tabla 4.21. Resultados para el cálculo de McNemar de Bagging contra ECOC

El valor obtenido al aplicar la ecuación 3.16 es de 6.322, por lo que, de nuevo, *Bagging* es el mejor de los dos métodos comparados.

f) Uno Contra Todos / ECOC

257	14
11	18

Tabla 4.22. Resultados para el cálculo de McNemar de OAA contra ECOC

El valor obtenido al aplicar el test es de 0.16, por lo que, en este caso, se puede concluir que, sobre el dominio evaluado, ambos métodos son estadísticamente equivalentes.

En la Tabla 4.23. se muestra un resumen de los valores anteriormente detallados y en la Figura 4.6 se muestra la pantalla de la aplicación una vez han terminado los distintos procesos y se han calculado los valores necesarios para aplicar el test de McNemar.

	Red Simple	Bagging	OAA	ECOC
Red Simple		28,07	16,68	13,02
Bagging	28,07		5,041	6,322
OAA	16,68	5,041		0,16
ECOC	13,02	6,322	0,16	

Tabla 4.23. Resumen de resultados obtenidos al aplicar McNemar a los Conjuntos de Clasificadores

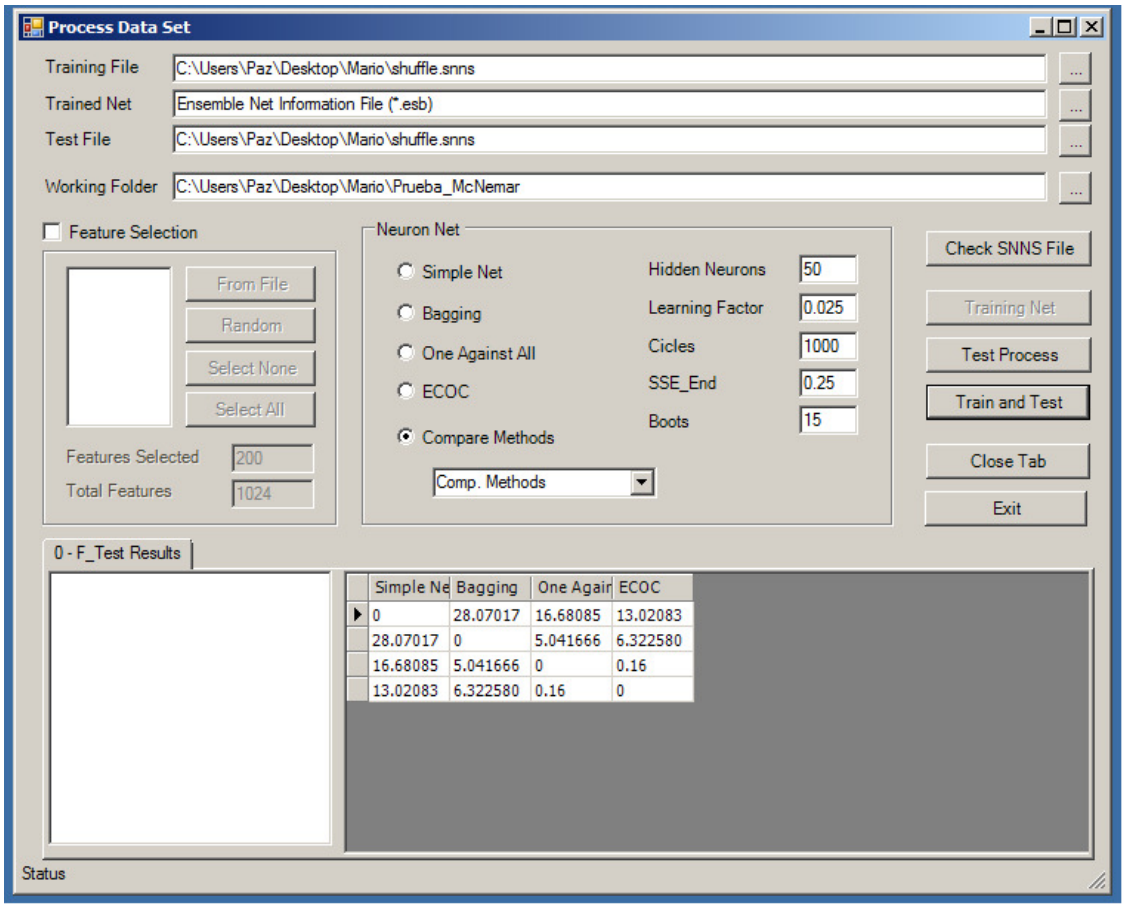


Figura 4.6. Proceso McNemar finalizado mostrando los resultados

Capítulo 5

Conclusiones y Trabajos Futuros

En este capítulo, en primer lugar, se resumen las principales dificultades encontradas durante el desarrollo de la aplicación. Después se extraen y exponen las conclusiones derivadas del proceso experimental. A continuación se muestran las líneas a seguir para nuevos trabajos, incluyendo el desarrollo de nuevas opciones, así como la mejora de algunos aspectos de rendimiento o visualización.

5.1. Dificultades encontradas

Entre todas las dificultades y problemas encontrados en la realización de este proyecto, destacan las siguientes:

- Generación aleatoria de redes iniciales

La creación de una red de neuronas requiere la inicialización aleatoria de los pesos asociados a cada interconexión. Estos pesos son ajustados durante el proceso de aprendizaje hasta conseguir que el comportamiento de la red se ajuste al esperado. En la fase de inicialización se detectó que todas las interconexiones tenían un peso de igual valor y como consecuencia la red nunca llegaba al estado óptimo de equilibrio. Analizando el código se detectó que el generador de números aleatorios se inicializaba dentro de un bucle y se optó por realizar la inicialización en un único punto del programa y propagar el valor obtenido como un parámetro de aquellas funciones que lo requerían.

- Determinación de valores óptimos para los parámetros variables de la aplicación.

Mediante distintas pruebas se determina reducir el número de neuronas ocultas de 100 a 50 ya que con este número los resultados son mejores.

- Generación de ficheros de almacenamiento de las redes creadas

Para evitar problemas surgidos por la sobre-escritura de distintos ficheros generados por la aplicación, se ha tenido que tratar cuidadosamente el nombre asignado y la carpeta en la que se almacenan.

Para facilitar la fase de evaluación, (recuperar las redes generadas para cada modelo) ha sido necesario generar un fichero en el que se almacena la información básica de cada conjunto de clasificadores (arquitectura empleada, número de redes que lo componen, ficheros de red de neuronas del conjunto).

5.2. Conclusiones

A continuación se exponen las conclusiones extraídas tras analizar cuidadosamente los resultados obtenidos en la fase experimental expuesta en el capítulo 4.

Analizando los valores obtenidos tras aplicar el Test de McNemar y el F_Test se puede deducir que las conclusiones derivadas de uno y otro no siempre son equivalentes. Así, mientras que para el F-test sólo hay diferencias significativas entre Bagging y la Red de Neuronas simple, al aplicar el test de McNemar se concluye que Bagging es mejor que el resto de sistemas analizados. Esta observación indica que los resultados derivados de la comparación estadística no siempre son concluyentes.

Observando la precisión de los distintos modelos de clasificación evaluados se puede comprobar que, tal y como se esperaba, los conjuntos de clasificadores parecen ser más precisos que el clasificador simple. Sin embargo, esta mejora en la precisión lleva asociada un gran incremento en el coste computacional de los sistemas. Por tanto, el uso de los conjuntos de clasificadores debería estar supeditado al compromiso precisión/coste.

En el estudio realizado, la reducción en el número de características lleva asociada una drástica disminución de la precisión. Con este dato no se pretende refutar la utilidad de la selección de características sino recalcar la necesidad de utilizar métodos de selección de características más elaborados que el usado en la realización de este proyecto (selección aleatoria). No obstante los resultados observados al implementar Bagging con y sin selección de características parecen indicar que este método es fiable incluso cuando se usa un conjunto de características que no es óptimo.

5.3. Trabajos Futuros

A continuación se exponen varias propuestas para continuar la línea de desarrollo de este proyecto.

1. Incluir nuevos métodos de clasificación. Por un lado se añadirían los procedimientos necesarios para la construcción de nuevos modelos de conjuntos de clasificadores como pudiera ser Boosting, y por otro lado se podrían incluir opciones para construir los clasificadores base de distintos tipos aparte de la Redes de Neuronas, como pudieran ser los Árboles de Decisión.

2. Implementar los algoritmos e incluir las opciones para medir la diversidad de los clasificadores construidos.

3. Añadir nuevos métodos de integración y dar la posibilidad al usuario de elegir uno de ellos para determinar la selección de la salida de los distintos conjuntos de clasificadores. Actualmente estos métodos están predefinidos para cada sistema según la documentación consultada.

4. Desarrollar algoritmos de selección de características.

5. Revisar los procesos de la aplicación para incluir programación multitarea (*threads*) con el objetivo de mejorar el rendimiento de la aplicación.

6. Añadir barras de progreso y ventanas de información del estado de los procesos para proveer al usuario de una información más detallada en cada momento de la situación de la creación de una red, entrenamiento, etc.

7. Permitir que la topología y la arquitectura de las redes de neuronas sea variable dentro del conjunto, es decir, permitir que las redes de neuronas de un conjunto de clasificadores tengan distinto número de neuronas ocultas, diferente coeficiente de aprendizaje, etc.

8. Modificar la estructura de la aplicación para trabajar con datos almacenados en bases de datos, con el objetivo de mejorar la eficiencia en la ejecución de los procesos y la gestión de los recursos. Este punto cuenta con el inconveniente de la necesidad de un servidor de base de datos y su correspondiente espacio de almacenamiento, así como el mantenimiento de la misma. La actual aplicación es fácil de instalar y no requiere mantenimiento ni espacio, salvo el necesario para los ficheros generados en las ejecuciones.

Capítulo 6

Presupuesto y Planificación del Proyecto

6.1. Planificación del Proyecto

La duración planificada del proyecto fue de 207.25 días. Se planificó de tal manera para poder presentarlo en el mes de octubre de 2011.

La planificación se basa en una jornada de trabajo de 4 horas, con 5 días laborables por semana, es decir, 20 horas semanales, con un Analista Programador y un Jefe de Proyecto que aprueba y supervisa cada una de las tareas definidas.

Los recursos utilizados son un ordenador personal con sistema operativo Windows XP, Microsoft Visual Studio 2003 y Microsoft Office 2003.

A continuación se detalla el diagrama de *Gantt* correspondiente a la planificación realizada para el proyecto:

ID	Task Name	Duration
1	Redacción de la Memoria	207.25 days
2	**FORMACIÓN**	**90 days**
3	C#	10 days
4	Redes de Neuronas y algoritomos de aprendizaje	40 days
5	Arquitecturas de Clasificadores	40 days
6	**ANALISYS Y DISEÑO**	**24 days**
7	Análisis de las estructuras de datos	4 days
8	Diseño de las clases	10 days
9	Diseño de los algoritmos	10 days
10	**DESARROLLO DE LA APLICACIÓN**	**123.5 days**
11	Diseño pantalla principal	3 days
12	Procedimientos para la lectura y gestión de los ficheros SNNS	4 days
13	Métodos para gestión y aplicación Selección de Atributos	4 days
14	**Implementación Neurona**	**8 days**
15	Neurona	2 days
16	Neurona Oculta	4 days
17	Neurona de Salida	2 days
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day

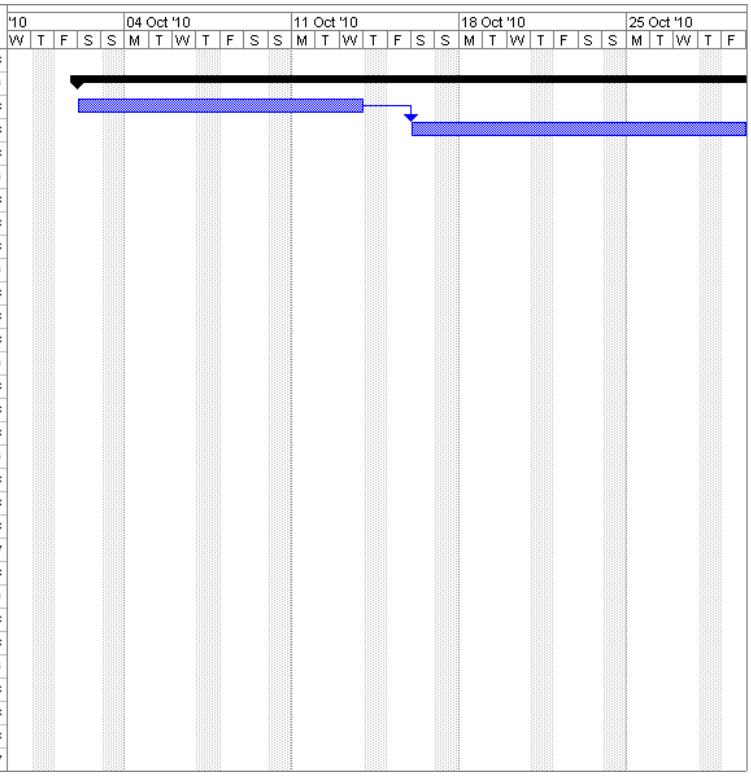


Figura 6.1. Planificación Octubre 2010

ID	Task Name	Duration
1	Redacción de la Memoria	207.25 days
2	**FORMACIÓN**	**90 days**
3	C#	10 days
4	Redes de Neuronas y algoritomos de aprendizaje	40 days
5	Arquitecturas de Clasificadores	40 days
6	**ANALISYS Y DISEÑO**	**24 days**
7	Análisis de las estructuras de datos	4 days
8	Diseño de las clases	10 days
9	Diseño de los algoritmos	10 days
10	**DESARROLLO DE LA APLICACIÓN**	**123.5 days**
11	Diseño pantalla principal	3 days
12	Procedimientos para la lectura y gestión de los ficheros SNNS	4 days
13	Métodos para gestión y aplicación Selección de Atributos	4 days
14	**Implementación Neurona**	**8 days**
15	Neurona	2 days
16	Neurona Oculta	4 days
17	Neurona de Salida	2 days
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day

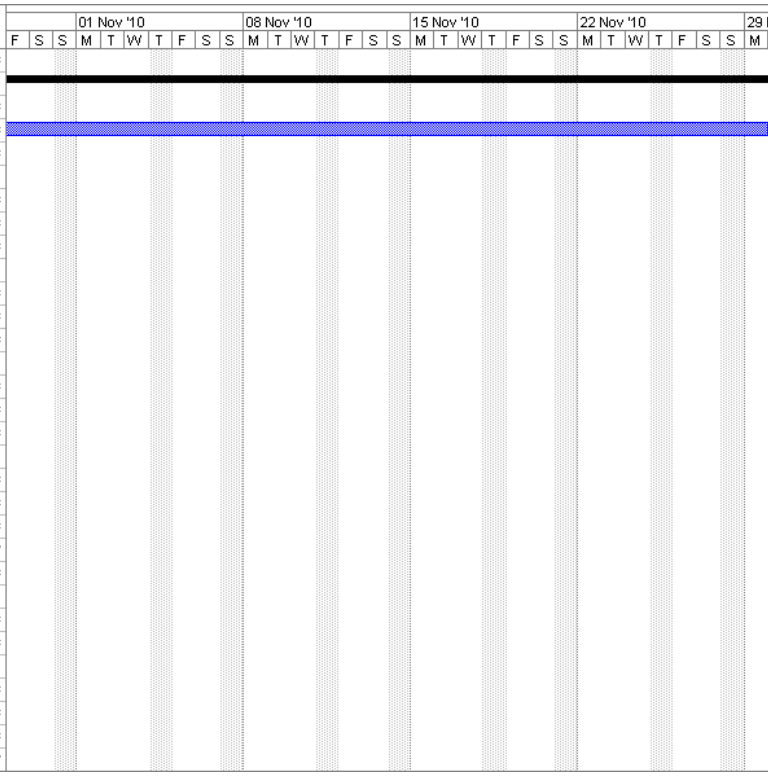


Figura 6.2. Planificación Noviembre 2010

ID	Task Name	Duration
1	Redacción de la Memoria	207.25 days
2	**FORMACIÓN**	**90 days**
3	C#	10 days
4	Redes de Neuronas y algoritomos de aprendizaje	40 days
5	Arquitecturas de Clasificadores	40 days
6	**ANALISYS Y DISEÑO**	**24 days**
7	Análisis de las estructuras de datos	4 days
8	Diseño de las clases	10 days
9	Diseño de los algoritmos	10 days
10	**DESARROLLO DE LA APLICACIÓN**	**123.5 days**
11	Diseño pantalla principal	3 days
12	Procedimientos para la lectura y gestión de los ficheros SNNS	4 days
13	Métodos para gestión y aplicación Selección de Atributos	4 days
14	**Implementación Neurona**	**8 days**
15	Neurona	2 days
16	Neurona Oculta	4 days
17	Neurona de Salida	2 days
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day

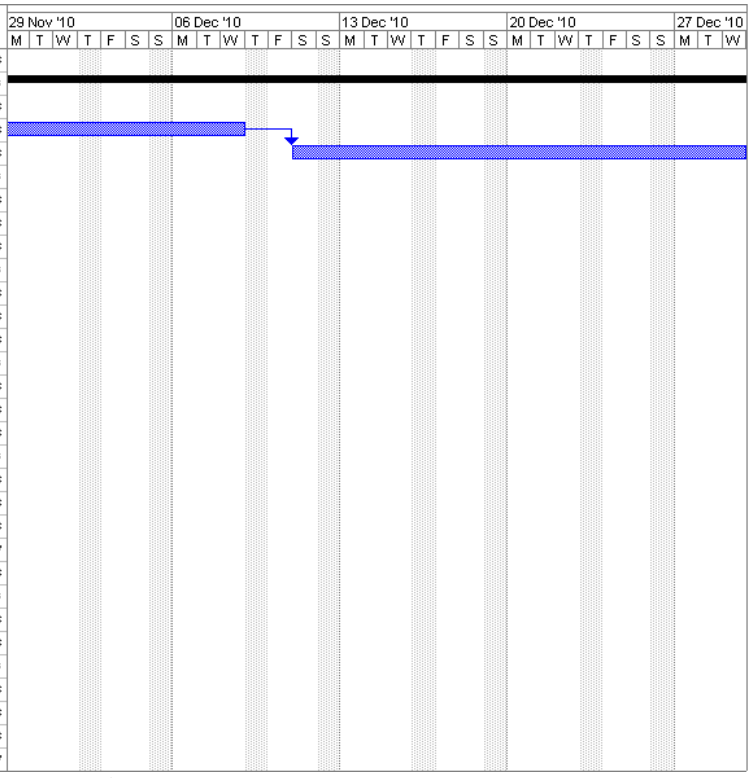


Figura 6.3. Planificación Diciembre 2010

ID	Task Name	Duration
1	Redacción de la Memoria	207.25 days
2	**FORMACIÓN**	**90 days**
3	C#	10 days
4	Redes de Neuronas y algoritomos de aprendizaje	40 days
5	Arquitecturas de Clasificadores	40 days
6	**ANALISYS Y DISEÑO**	**24 days**
7	Análisis de las estructuras de datos	4 days
8	Diseño de las clases	10 days
9	Diseño de los algoritmos	10 days
10	**DESARROLLO DE LA APLICACIÓN**	**123.5 days**
11	Diseño pantalla principal	3 days
12	Procedimientos para la lectura y gestión de los ficheros SNNS	4 days
13	Métodos para gestión y aplicación Selección de Atributos	4 days
14	**Implementación Neurona**	**8 days**
15	Neurona	2 days
16	Neurona Oculta	4 days
17	Neurona de Salida	2 days
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day

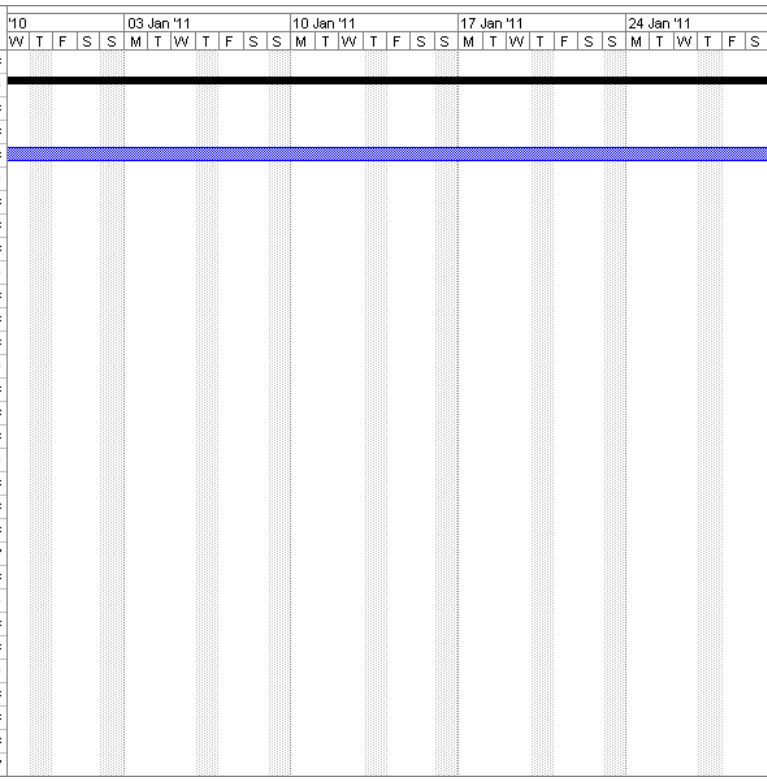


Figura 6.4. Planificación Enero 2011

ID	Task Name	Duration
1	Redacción de la Memoria	207.25 days
2	**FORMACIÓN**	**90 days**
3	C#	10 days
4	Redes de Neuronas y algoritomos de aprendizaje	40 days
5	Arquitecturas de Clasificadores	40 days
6	**ANALISYS Y DISEÑO**	**24 days**
7	Análisis de las estructuras de datos	4 days
8	Diseño de las clases	10 days
9	Diseño de los algoritmos	10 days
10	**DESARROLLO DE LA APLICACIÓN**	**123.5 days**
11	Diseño pantalla principal	3 days
12	Procedimientos para la lectura y gestión de los ficheros SNNS	4 days
13	Métodos para gestión y aplicación Selección de Atributos	4 days
14	**Implementación Neurona**	**8 days**
15	Neurona	2 days
16	Neurona Oculta	4 days
17	Neurona de Salida	2 days
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day

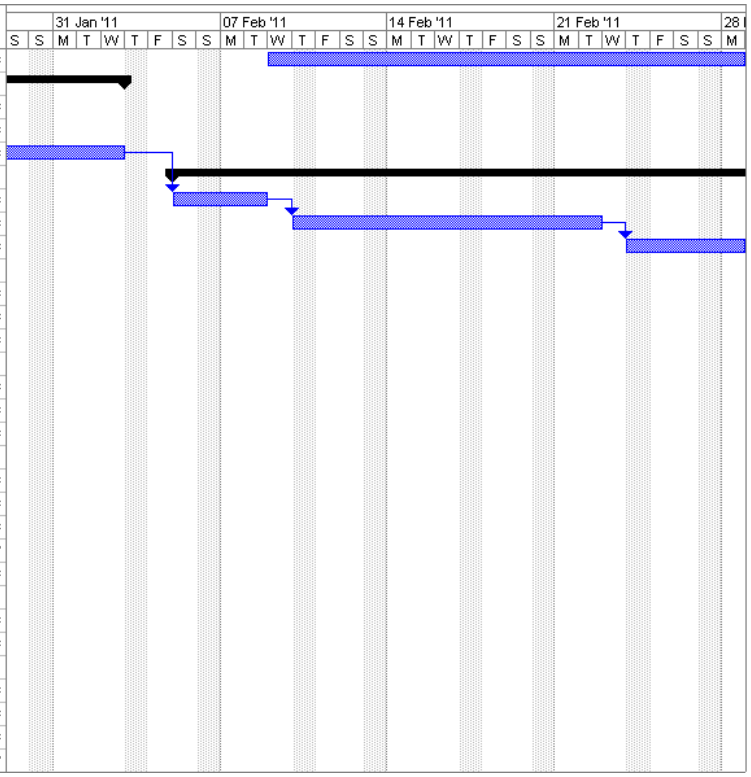


Figura 6.5. Planificación Febrero 2011

ID	Task Name	Duration
1	Redacción de la Memoria	207.25 days
2	**FORMACIÓN**	**90 days**
3	C#	10 days
4	Redes de Neuronas y algoritomos de aprendizaje	40 days
5	Arquitecturas de Clasificadores	40 days
6	**ANALISYS Y DISEÑO**	**24 days**
7	Análisis de las estructuras de datos	4 days
8	Diseño de las clases	10 days
9	Diseño de los algoritmos	10 days
10	**DESARROLLO DE LA APLICACIÓN**	**123.5 days**
11	Diseño pantalla principal	3 days
12	Procedimientos para la lectura y gestión de los ficheros SNNS	4 days
13	Métodos para gestión y aplicación Selección de Atributos	4 days
14	**Implementación Neurona**	**8 days**
15	Neurona	2 days
16	Neurona Oculta	4 days
17	Neurona de Salida	2 days
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day

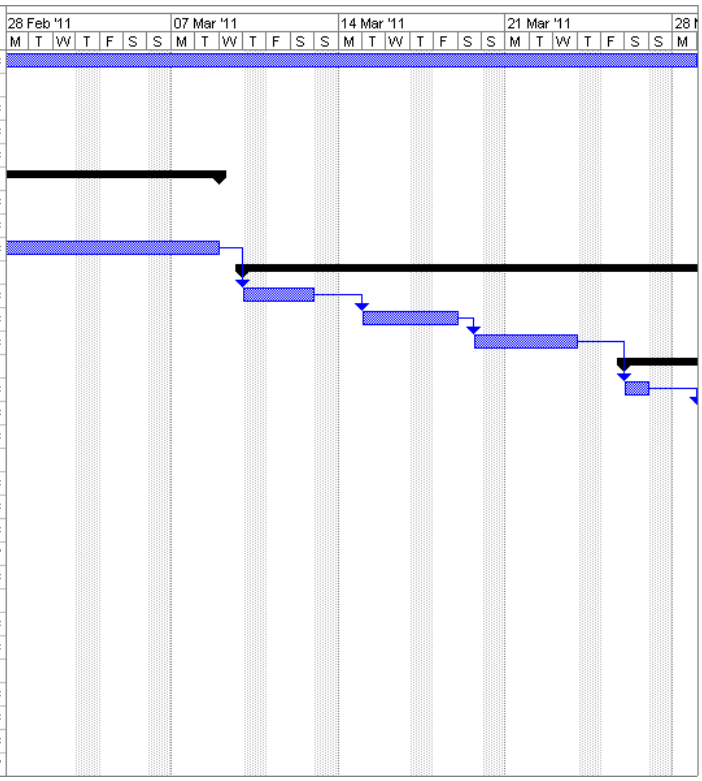


Figura 6.6. Planificación Marzo 2011

ID	Task Name	Duration
10	**DESARROLLO DE LA APLICACIÓN**	**123.5 days**
11	Diseño pantalla principal	3 days
12	Procedimientos para la lectura y gestión de los ficheros SNNS	4 days
13	Métodos para gestión y aplicación Selección de Atributos	4 days
14	**Implementación Neurona**	**8 days**
15	Neurona	2 days
16	Neurona Oculta	4 days
17	Neurona de Salida	2 days
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day
32	Pruebas de sistema y correcón de fallos	2 days
33	**Uno Contra Todos (OAA)**	**6 days**
34	Creación de las redes y entrenamiento	2 days
35	Función de combinación	1.5 days
36	Proceso de test	1 day
37	Pruebas de sistema y corrección de fallos	1.5 days
38	**ECOC**	**18.5 days**
39	Métodos para carga y gestión de la matriz de códigos ECO	2 days
40	Generación de subconjuntos ECOC	3 days

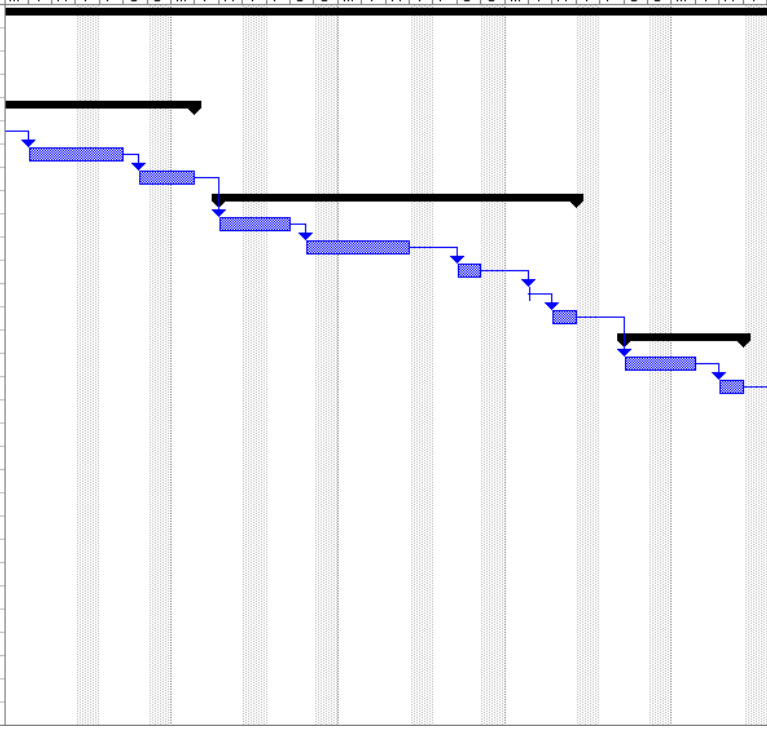


Figura 6.7. Planificación Abril 2011

ID	Task Name	Duration
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day
32	Pruebas de sistema y correcón de fallos	2 days
33	**Uno Contra Todos (OAA)**	**6 days**
34	Creación de las redes y entrenamiento	2 days
35	Función de combinación	1.5 days
36	Proceso de test	1 day
37	Pruebas de sistema y corrección de fallos	1.5 days
38	**ECOC**	**18.5 days**
39	Métodos para carga y gestión de la matriz de códigos ECO	2 days
40	Generación de subconjuntos ECOC	3 days
41	Creación de Redes y entrenamiento	4 days
42	Función de Combinación (distancia Hamming)	3.5 days
43	Proceso de test	2 days
44	Pruebas de sistema y correción de fallos	4 days
45	**Cross Validation 5x2cv**	**5.5 days**
46	División ficheros balanceados Folds	1.5 days
47	Desarrollo iteraciones y llamadas a los distintos métodos	4 days
48	**F_Test**	**11 days**

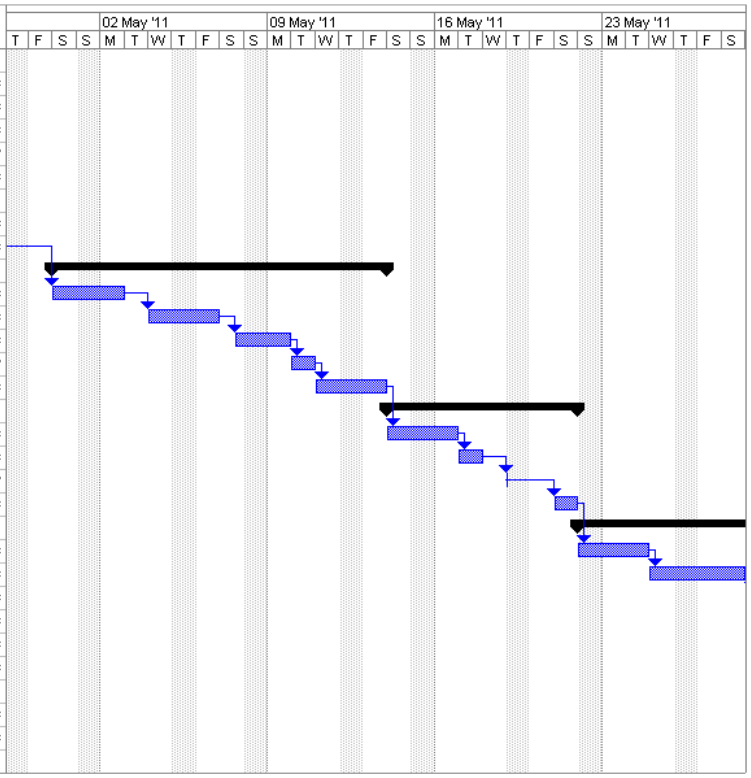


Figura 6.8. Planificación Mayo 2011

ID	Task Name	Duration
30	Función de combinación	1.5 days
31	Proceso de test	1 day
32	Pruebas de sistema y correcón de fallos	2 days
33	**Uno Contra Todos (OAA)**	**6 days**
34	Creación de las redes y entrenamiento	2 days
35	Función de combinación	1.5 days
36	Proceso de test	1 day
37	Pruebas de sistema y corrección de fallos	1.5 days
38	**ECOC**	**18.5 days**
39	Métodos para carga y gestión de la matriz de códigos ECO	2 days
40	Generación de subconjuntos ECOC	3 days
41	Creación de Redes y entrenamiento	4 days
42	Función de Combinación (distancia Hamming)	3.5 days
43	Proceso de test	2 days
44	Pruebas de sistema y correción de fallos	4 days
45	**Cross Validation 5x2cv**	**5.5 days**
46	División ficheros balanceados Folds	1.5 days
47	Desarrollo iteraciones y llamadas a los distintos métodos	4 days
48	**F_Test**	**11 days**
49	Implementación función F_Test	2 days
50	Implantación con 5x2cv	4 days
51	Pruebas de sistema y corrección de fallos	5 days
52	**McNemar**	**8 days**
53	Selección de ficheros de entrenamiento y test	1.5 days
54	Llamada a los distintos métodos	1.5 days
55	Implementación cálculo McNemar	2 days
56	Pruebas de sistema y corrección de fallos	3 days
57	**PRUEBAS**	**28 days**
58	Detección de problemas en la aplicación	5 days
59	Solución de los problemas encontrados	6 days
60	Análisis de los datos y detección de problemas	5 days

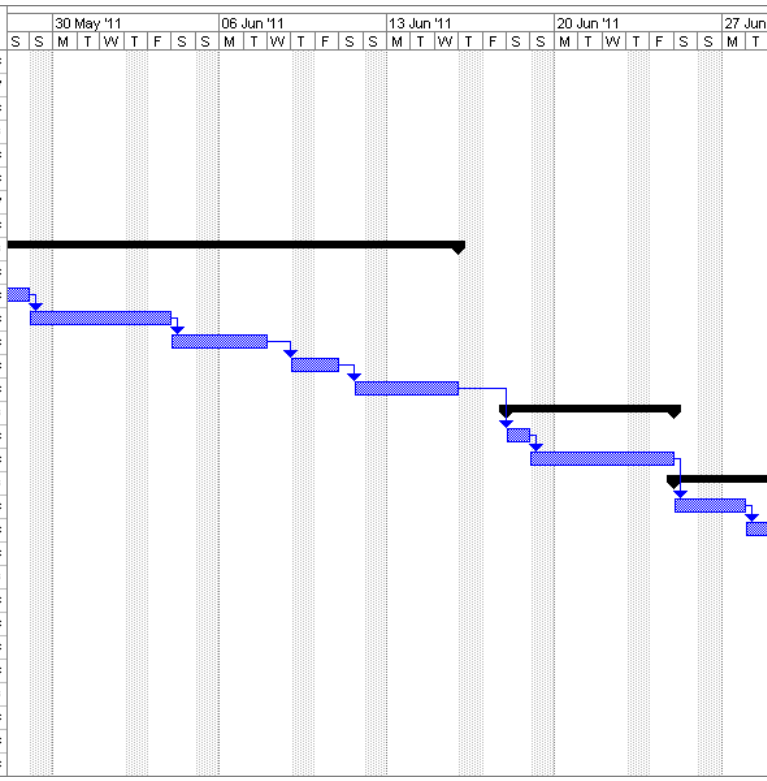


Figura 6.9. Planificación Junio 2011

ID	Task Name	Duration
45	**Cross Validation 5x2cv**	**5.5 days**
46	División ficheros balanceados Folds	1.5 days
47	Desarrollo iteraciones y llamadas a los distintos métodos	4 days
48	**F_Test**	**11 days**
49	Implementación función F_Test	2 days
50	Implantación con 5x2cv	4 days
51	Pruebas de sistema y corrección de fallos	5 days
52	**McNemar**	**8 days**
53	Selección de ficheros de entrenamiento y test	1.5 days
54	Llamada a los distintos métodos	1.5 days
55	Implementación cálculo McNemar	2 days
56	Pruebas de sistema y corrección de fallos	3 days
57	**PRUEBAS**	**28 days**
58	Detección de problemas en la aplicación	5 days
59	Solución de los problemas encontrados	6 days
60	Análisis de los datos y detección de problemas	5 days
61	Resolución de los problemas encontrados	6 days
62	Pruebas de determinación de las variables óptimas	6 days
63	Aprobacíon de la calidad de la aplicación y los datos	0 days
64	**FASE EXPERIMENTAL**	**13.5 days**
65	**Red de Neuronas simple**	**1 day**
66	Entrenamiento y test red de Neuronas	0.5 days
67	Análsis de resultados	0.5 days
68	**Baggin**	**1 day**
69	Entrenamiento y test Bagging	0.5 days
70	Análsis de resultados	0.5 days
71	**OAA**	**1 day**
72	Entrenamiento y test OAA	0.5 days
73	Análsis de resultados	0.5 days
74	**ECOC**	**1.5 days**
75	Entrenamiento y test ECOC	1 day

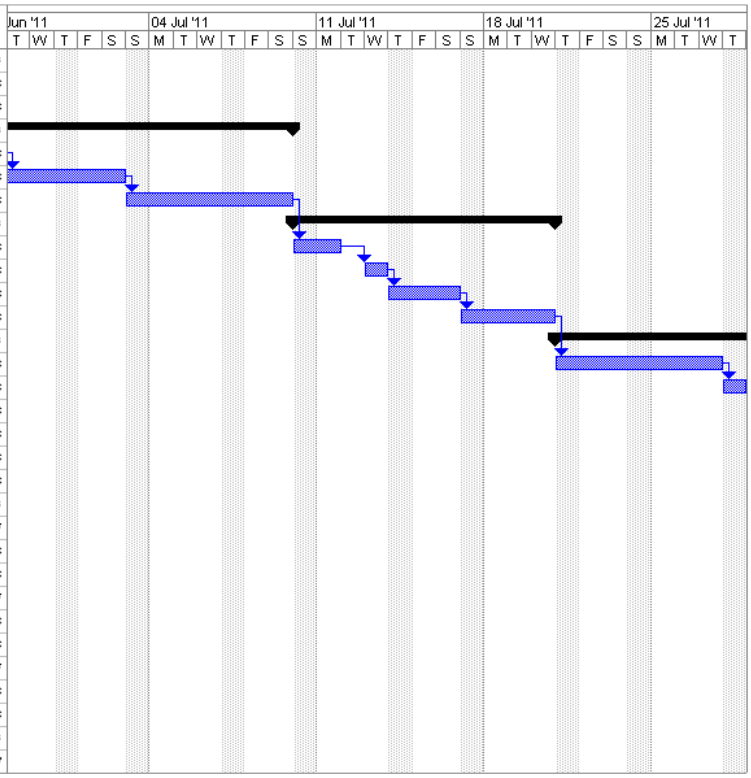


Figura 6.10. Planificación Julio 2011

ID	Task Name	Duration
57	**PRUEBAS**	**28 days**
58	Detección de problemas en la aplicación	5 days
59	Solución de los problemas encontrados	6 days
60	Análisis de los datos y detección de problemas	5 days
61	Resolución de los problemas encontrados	6 days
62	Pruebas de determinación de las variables óptimas	6 days
63	Aprobacíon de la calidad de la aplicación y los datos	0 days
64	**FASE EXPERIMENTAL**	**13.5 days**
65	**Red de Neuronas simple**	**1 day**
66	Entrenamiento y test red de Neuronas	0.5 days
67	Análsis de resultados	0.5 days
68	**Baggin**	**1 day**
69	Entrenamiento y test Bagging	0.5 days
70	Análsis de resultados	0.5 days
71	**OAA**	**1 day**
72	Entrenamiento y test OAA	0.5 days
73	Análsis de resultados	0.5 days
74	**ECOC**	**1.5 days**
75	Entrenamiento y test ECOC	1 day
76	Análsis de resultados	0.5 days
77	**F_Test 5x2cv**	**4.5 days**
78	Ejecución proceso Cross Validation y cálculo F_Test	4 days
79	Análsis de resultados	0.5 days
80	**McNemar**	**3.5 days**
81	Obtención de resultados pruebas y cálculo de McNemar	3 days
82	Análsis de resultados	0.5 days
83	Determinación y redacción de conclusiones	4 days

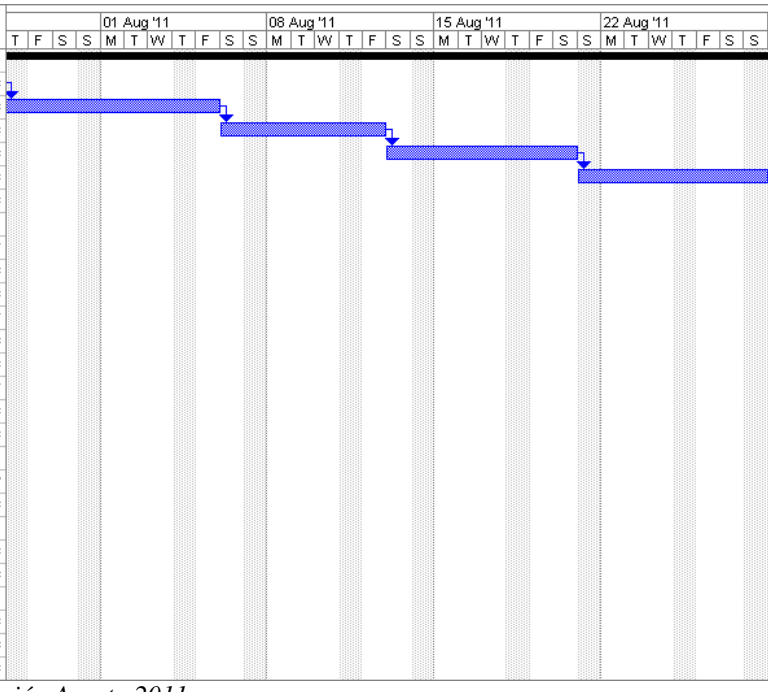


Figura 6.11. Planificación Agosto 2011

ID	Task Name	Duration
58	Detección de problemas en la aplicación	5 days
59	Solución de los problemas encontrados	6 days
60	Análisis de los datos y detección de problemas	5 days
61	Resolución de los problemas encontrados	6 days
62	Pruebas de determinación de las variables óptimas	6 days
63	Aprobacíon de la calidad de la aplicación y los datos	0 days
64	**FASE EXPERIMENTAL**	**13.5 days**
65	**Red de Neuronas simple**	**1 day**
66	Entrenamiento y test red de Neuronas	0.5 days
67	Análsis de resultados	0.5 days
68	**Baggin**	**1 day**
69	Entrenamiento y test Bagging	0.5 days
70	Análsis de resultados	0.5 days
71	**OAA**	**1 day**
72	Entrenamiento y test OAA	0.5 days
73	Análsis de resultados	0.5 days
74	**ECOC**	**1.5 days**
75	Entrenamiento y test ECOC	1 day
76	Análsis de resultados	0.5 days
77	**F_Test 5x2cv**	**4.5 days**
78	Ejecución proceso Cross Validation y cálculo F_Test	4 days
79	Análsis de resultados	0.5 days
80	**McNemar**	**3.5 days**
81	Obtención de resultados pruebas y cálculo de McNemar	3 days
82	Análsis de resultados	0.5 days
83	Determinación y redacción de conclusiones	4 days

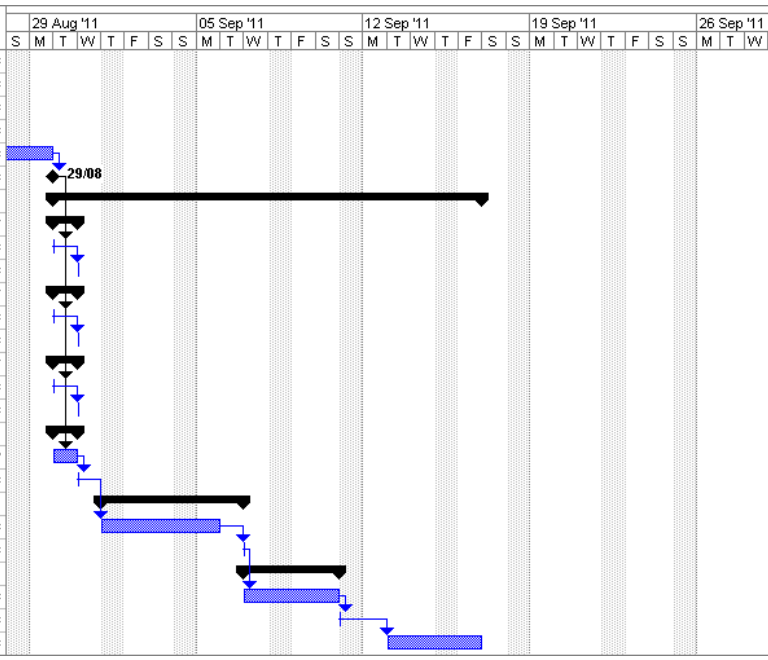


Figura 6.12. Planificación Septiembre 2011

ID	Task Name	Duration
1	Redacción de la Memoria	207.25 days
2	**FORMACIÓN**	**90 days**
3	C#	10 days
4	Redes de Neuronas y algoritomos de aprendizaje	40 days
5	Arquitecturas de Clasificadores	40 days
6	**ANALISYS Y DISEÑO**	**24 days**
7	Análisis de las estructuras de datos	4 days
8	Diseño de las clases	10 days
9	Diseño de los algoritmos	10 days
10	**DESARROLLO DE LA APLICACIÓN**	**123.5 days**
11	Diseño pantalla principal	3 days
12	Procedimientos para la lectura y gestión de los ficheros SNNS	4 days
13	Métodos para gestión y aplicación Selección de Atributos	4 days
14	**Implementación Neurona**	**8 days**
15	Neurona	2 days
16	Neurona Oculta	4 days
17	Neurona de Salida	2 days
18	**Implementación Red de Neuronas**	**12 days**
19	Algoritmo BackPropagation	3 days
20	Implementación entrenamiento Red de Neuronas	4 days
21	Implementación Tests de Redes de Neuronas	2 days
22	Pruebas de sistema contra procesos SNNS	1 day
23	Resolución problemas encontrados	2 days
24	**Matriz de Confusión**	**5 days**
25	Diseño pantalla Matriz de Confusión	3 days
26	Implementación métodos Matriz	2 days
27	**Bagging**	**10.5 days**
28	Métodos para creación de Boots	3 days
29	Creación de las redes y entrenamiento	3 days
30	Función de combinación	1.5 days
31	Proceso de test	1 day

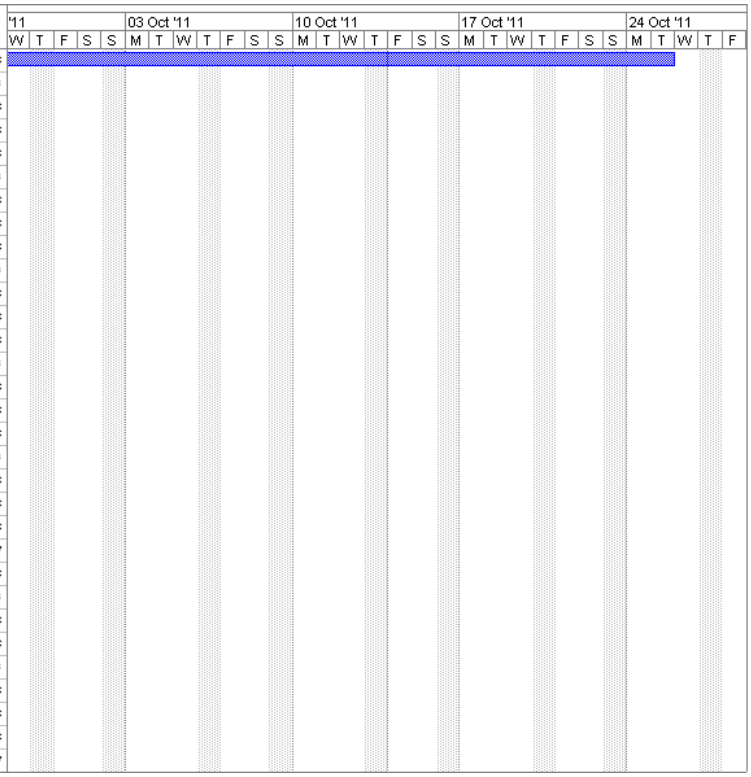


Figura 6.13. Planificación Octubre 2011

6.2. Presupuesto

En base a la planificación realizada en el apartado anterior con una duración del proyecto de 207.25 días, sumando un total de 829 horas, se obtiene el presupuesto detallado a continuación:

El coste de personal constituye la mayor proporción del coste total del proyecto. Se entiende por coste de personal aquellos derivados de la dedicación de los trabajadores a las distintas actividades del proyecto. El resto de costes, que intervienen en menor medida, se detallan posteriormente.

El primer paso en el cálculo de costes de personal es determinar el coste/hora de cada uno de los profesionales que participan en el desarrollo del sistema. En este proyecto se considera la participación de los perfiles que aparecen en la siguiente tabla, junto a su correspondiente sueldo por hora.

Categoría	€ / hora
Jefe de Proyecto	80
Analista Programador	50

Tabla 6.1. Costes de personal

A continuación se calcula el coste total de personal según las horas resultantes de la estimación detallada en el apartado anterior, como se muestra en la tabla 6.2.

Categoría	Horas totales	Coste total
Jefe de Proyecto	829	66.320
Analista Programador		41.450
	Total Coste de personal	**107.770**

Tabla 6.2. Cálculo del total de Costes de personal

Debido a los rápidos avances tecnológicos en hardware y software, estos recursos han de ser renovados periódicamente, por lo que se intenta establecer una relación calidad/tiempo de amortización óptima. La vida útil de los equipos informáticos es 3 años, por lo que se amortiza un tercio de su coste anualmente.

Los costes generados en concepto de recursos informáticos empleados en la realización del proyecto, se presentan en la tabla 6.3.

Concepto	€
Ordenador personal	867
Monitor 19”	112
Conexión ADSL	64 €/mes * 12 meses = 768
Total Coste de Material	**1.747**

Tabla 6.3. Cálculo del total de Costes de material

La parte de amortización imputada al proyecto es la proporcional a la duración del mismo, que según los cálculos del apartado anterior puede estimarse en alrededor de 12 meses dedicados al proyecto en cuestión –considerando un mes con 20 días de 4 horas de trabajo cada uno– (aunque se distribuyan a lo largo de 12 meses naturales). Así, los 12 meses sobre los 36 de vida útil, arrojan la parte de coste de equipos que es amortizada durante la realización de este proyecto de 1747 * (12/36) = **582,33** €

Para la realización del proyecto se utilizan los productos software indicados en la tabla 6.4 con información de sus costes de licencia

Producto	€
Licencia Microsoft Visual Studio	653,93
Licencia Microsoft Office Student	79,95
Coste Total Licencias Software	**733,88**

Tabla 6.4. Cálculo del total de Costes de licencias software

Los gastos indirectos del proyecto se calculan como el 5 % sobre el total del resto de costes. En este proyecto, este apartado incluye fundamentalmente conceptos como los gastos ocasionados por el consumo eléctrico, con lo que el gasto total de este proyecto, quedaría como se puede ver en la tabla 6.5

Concepto	€
Personal	107.770
Material	582,33
Licencias SW	733,88
Gastos Indirectos (5 %)	5.454,31
Coste Total	**114.540,52**

Tabla 6.5. Cálculo del coste total del proyecto

Dada la naturaleza y objetivo de éste proyecto no se considera necesario ni oportuno incluir ningún porcentaje de beneficio sobre el coste total del proyecto, aunque formarían parte de un presupuesto a presentar a un cliente. Tampoco se ha tenido en cuenta el IVA en los costes presentados.

Así, el coste total del proyecto presentado en esta propuesta es **114.540,52 €.**

Referencias

[Alpaydin, 1999] E. Alpaydin. Combined 5x2cv F test for comparing supervised learning algorithms, *Neural Computation*, vol. 11, 1885-1892. 1999.

[Battiti & Colla, 1994] R. Battiti and A. Colla. Democracy in neural nets: Voting schemes for classification. *Neural Networks*, 7(4), 691–707, 1994

[Breiman, 1996] L. Breiman, Bagging Predictors, *Machine Learning*, 24, 123-140, 1996

[Brill et al., 2003] R. Brill, R. Gutierrez-Osuna, F. Quek, Atribute Bagging: improving accuracy of classifier esenbles by using random feature subsets. *Pattern Recognition 36*, 1291-1302, 2003

[Cantú-Paz & Kamath, 1995] E. Cantú-Paz, C. Kamath, "An Empirical Comparison of Combinatios of Evolutionary Algorithms and Neural Networks for Classification Problems", *IEEE Transactions on systems, Man and Cybernetics – Part B: Cybernetics,* 915-927. 2005.

[Dietterich & Bakiri, 1991] T.G. Dietterich and G. Bakiri. Error-correcting output codes: a general method for improving multiclass inductive learning programs. *In T. L. Dean and K. McKeown, editors, Proceedings of the Ninth AAAI National Conference on Artificial Intelligence,* 572–577, Menlo Park, CA, 1991. AAAI Press.

[Dietterich & Bakiri, 1995] T.G. Dietterich and G. Bakiri. Solving Multiclass Learning Problemas via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research* 2, 263-286, 195

[Dietterich, 1997] T. G. Dietterich. Machine Learning research: four currents directions. *AI Magazine*, 18(4):97-136, 1997.

[Dietterich, 1998] T. G. Dietterich, Approximate statistical test for comparing supervised classification learning algorithms, *Neural Computation* 10, 1895-1923, 1998

[Dietterich, 2000] T.G. Dietterich,: An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning,* vol. 40, 2, 139-157, 2000.

[Giacinto & Roli, 2001] G. Giacinto and F. Roli. An approach to the automatic design of multiple classifier systems. Pattern Recogn. Lett., 22, 25–33, 2001.

[Freund & Schapire, 1996] Y. Freund, R.E. Schapire. Experiments with a new boosting algorithm. Proceedings *13th International Conference on Machine Learning*, 148-156, 1996.

[Guyon & Elisseeff, 2003] An Introduction to Variable and Feature Selection. *Journal of Machine Research 3*, 1157-1182, 2003

[Hall, 1999] M. A. Hall, Correlation-based Feature Selection for Machine Learning. *Department of Computer Science, Univ. of Waikato*.1999

[Hansen & Salamon, 1990] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001, 1990.

[Herrera et al., 2004] F. Herrera, C. Hervás, J. Otero, L. Sánchez. Un estudio empírico preliminar sobre los tests estadísticos más habituales en el aprendizaje automático, R. Giraldez, J.C. Riquelme, J.S. Aguilar (Eds.) *Tendencias de la Minería de Datos en España, Red Española de Minería de Datos y Aprendizaje* (TIC2002-11124-E). pp. 403-412. 2004.

[Hernández-Orallo et al., 2007] J. Hernández-Orallo, M.J. Ramírez, C. Ferri. Introducción a la minería de datos. *Pearson Educación, S.A*., Madrid, 2004.

[Isasi & Galván, 2004] P. Isasi, I. M. Galván. Redes de Neuronas Artificiales. Un Enfoque Práctico. *Pearson Educación S.A.* 2004

[Krogh & Vedelsby, 1995] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in* Neural Information Processing Systems, volume 7, pages 231–238. The MIT Press, 1995.

[Kuncheva, 2002] L. I. Kuncheva. Switching Between Selection and Fusion in Combining Classifiers: an Experiment. *IEEE Transactions on Systems, Man and Cybernetics* Vol 32, Nº 2, 146-156

[Kuncheva & Whitaker, 2002] L.I. Kuncheva and C.J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning,* 51(2):181–207, 2002.

[Kuncheva & Whitaker, 2003] L.I. Kuncheva, C.J. Whitaker. Measures of diversity in classifier ensembles, *Machine Learning* , 51 , 2003, 181-207

[McCulloch & Pitts, 1943] W. S. McCulloch y W. Pitts. A logical calculus of the ideas immanent in neurons activity, Bull. Math. Biophys., 5, 115-133, (1943).

[McNemar, 1947] Q. McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12, 153-157, 1947

[Opitz, 1999] D. W. Opitz, Feature Selection for Esembles, *AAAI-99 Procedings*, 1999

[Parmanto et al., 1996] Parmanto B, Munro PW, Doyle HR (1996) Improving committee diagnosis with resampling techniques. In *MIT Press, Cambridge*, pp 882–888, 1996

[Ranawana & Palade, 2005] R. Ranawana and V. Palade. Multi-Classifier Systems - A Review and Roadmap for Developers. University of Oxford Computing Laboratory, Oxford, United Kingdom, 2005

[Rifkin & Klautau, 2004] B. Rifkin and A. Klautau. In Defense of one-vs-all classification. *Jouranl of Machine learning Research*, 5, pp 101-141, 2004

[Rosenblatt, 1959] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 386-408. 1959

[Schapire, 1990] R. Schapire. The strength of weak learnability. Machine Learning, (2), 197-227. 1990.

[Sesmero, 2010] M. P. Sesmero, J. M. Alonso-Weber, G. Gutiérrez, A. Ledesma, A. Sanchos. A new Artificial Neural Network Ensemble Based on Feature Selection and Class Redoding. *Neural Comput. & Applic*. 2010

[Skurichina & Duin, 2001] M. Skurichina, Robert P. W. Duin. Bagging and the Random Subspace Method for Redundant Feature Spaces. *Multiple Classifier Systems*, pp.1-10. 2001

[Stone, 1978] M. Stone. Cross-validation: A review, *Matemastische Operationsforschung Statischen, Serie Statistics*, 9, pp. 127-139. 1978.

[Stoppiblia et al., 2003] Stoppiblia H., Dreyfus G., Debois R. & Oussar Y. 2003. Ranking a random feature for variable and feature selection. *Journal of Machine Learning Reseach*. Vol. 3 p. 1399-1414

[Tsymbal et al., 2005] A. Tsymbal, M. Pechemizkiy, P. Cunningham, Diversity in search strategies for ensemble feature selection, *Information Fusion 6* 83-98, 2005

[Wolpert, 1992] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[Zhu et al., 2004] X. Zhu, X. Wu, and Y. Yang. Dynamic Selection for effective mining from noisy data streams. P*roceedings of Fourth IEEE International Conference on Data Mining*, pp. 205-312

Apéndice A

MANUAL DE USUARIO

Este anexo amplía lo expuesto en el apartado 4.2 de este documento acerca de la interfaz de la aplicación, presentando un manual sobre el manejo de la misma.

En la parte superior, se encuentran las cajas de texto que indican la ruta de los ficheros de entrenamiento y pruebas, así como la carpeta de trabajo donde la aplicación creará los ficheros necesarios para crear y entrenar las redes de neuronas, y extraer los resultados de salida, como se detalla a continuación:

- *Training File*: Selección del fichero de entrenamiento para crear una Red de Neuronas
- *Trained Net*: Selección de una Red de Neuronas existente, previamente creada por la aplicación.
- *Test File*: Selección del fichero de test. Por defecto se rellena con el mismo fichero que el indicado en *Training File.*
- *Working Fólder*: Selección de el directorio donde se almacenarán los ficheros generados en los procesos de la aplicación. Por defecto toma el valor del directorio donde se encuentre el fichero de entrenamiento seleccionado.

Antes de comenzar cualquier proceso, deben ser seleccionados los ficheros de entrenamiento o pruebas, según corresponda, así como el fichero *esb* de configuración de *Trained Net* cuando se quiere trabajar con una Red existente, ya sea para la realizar de pruebas o para continuar con el entrenamiento.

En la parte izquierda se encuentra la parte que se refiere a la selección de características, Seleccionando esta opción, se activa la selección de atributos para el proceso que se vaya a comenzar. Se puede realizar la selección de los atributos de tres maneras:

- *From File:* Pulsando este botón se puede seleccionar un fichero con la selección a aplicar.

- *Random:* Se realiza una selección aleatoria de los elementos indicados en la caja de texto *Feature Selection*.
- *Select None / All:* Pulsando sobre estos botones o en la lista de la izquierda, se pueden seleccionar manualmente los atributos deseados.

En el centro de la pantalla se sitúan los distintos sistemas de conjuntos de clasificadores que podemos seleccionar para ejecutar un proceso:

- *Simple Net.* Seleccionando esta opción, el proceso trabajará con un conjunto de clasificadores de un solo clasificador y una única salida.
- *Bagging:* Con esta opción seleccionada se trabajará con la arquitectura *Bagging,* creándose tantos ficheros a partir del fichero de entrada y por lo tanto tantos clasificadores base como se indique en la caja de texto *Boots*
- *One Against All:* Los procesos emplearán el método de Uno Contra Todos si esta opción es seleccionada.
- *ECOC:* Si se selecciona esta opción, el sistema pedirá al usuario que seleccione el fichero de códigos *ECOC*, para construir, entrenar o probar esta arquitectura.
- *Compare Methods*: Seleccionando esta opción para la comparación de métodos se habilita la lista pata seleccionar el test estadístico con el que proceder:
    - *F_Test*: se realizará un proceso de Validación Cruzada de cada método para calcular los valores de F_Test resultado de comparar todos los anteriores modelos.
    - *McNemar.* Se realizará el cálculo de McNemar en base al proceso de entrenamiento y test de los módelos.

A la derecha de estas opciones, se encuentran las casillas para definir los parámetros necesarios para configurar una Red de Neuronas, un Conjunto de Clasificadores, proceso de aprendizaje, etc., es decir, los valores que necesitará el sistema para procesar la información. Las variables disponibles son:

- Número de neuronas ocultas que componen esta capa de la Red de Neuronas
- Coeficiente de aprendizaje, necesario en el algoritmo de retro-propagación (*back propagation*) para entrenar la red.
- Número de ciclos que realizará el algoritmo de aprendizaje en esta fase.
- Boots (solo para el método *Bagging*), número de subconjuntos en que será dividido el fichero para crear la arquitectura Bagging.
- SSE_End es un valor de referencia para el algoritmo de aprendizaje, que marca el límite en el cual finaliza el proceso.

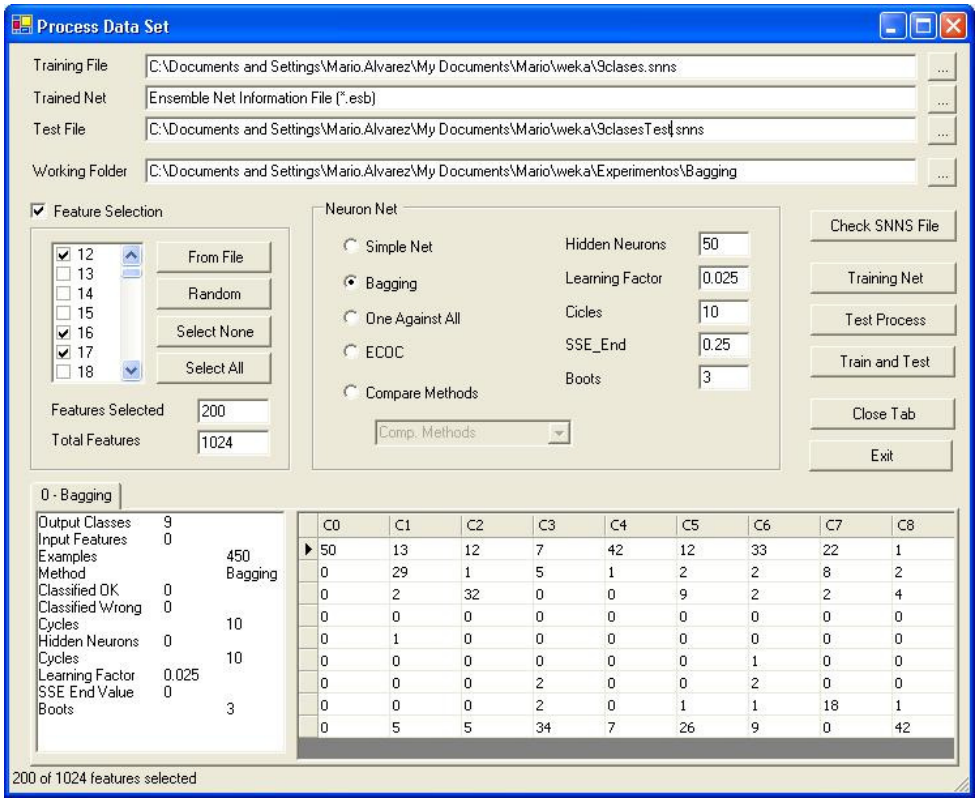
A la derecha de la pantalla, están los botones de proceso de la aplicación, que realizan las siguientes funciones:

- *Check SNNS*: Carga en memoria un fichero de ejemplos con formato SNNS y comprueba que es correcto
- *Training Net*: Crea y entrena un Conjunto de Clasificadores según el método y los valores indicados. Se solicita al usuario la ruta y nombre del fichero *esb* para guardar los datos de la red que es creado al finalizar el entrenamiento. Si el

método seleccionado es *ECOC*, se pide al usuario que seleccione el fichero que contenga la matriz de códigos *ECOC*. Este botón está inhabilitado cuando se selecciona la opción *Compare Methods.* Si se indica un fichero de red existente esb en la caja de texto correspondiente, el proceso carga esa red y continúa su entrenamiento. Si se ha seleccionado la selección de características, se generará un fichero con extensión *fs* donde se almacenará la información con el formato definido para los ficheros de selección de características.

- *Test Process:* Lanza un proceso de prueba con el fichero indicado y muestra los resultados en pantalla a través de la matriz de confusión que se añade en las pestañas de la parte inferior de la pantalla. Para ejecutar este proceso es necesario seleccionar un fichero esb de red existente que permitirá al sistema cargar la red correspondiente.
- *Train and Test:* Toma el fichero de entrenamiento seleccionado, crea y entrena un conjunto de clasificadores según la opción indicada, realiza la fase de pruebas con el fichero también seleccionado, y muestra los resultados en pantalla y en los ficheros de salida creados en la carpeta seleccionada.
- *Close Tab:* Cierra la pestaña activa de la parte inferior.
- *Exit:* Cierra la aplicación.

Por último, en la parte inferior del interfaz se encuentran las tablas de resultados, donde se van añadiendo en nuevas pestañas los resultados obtenidos al ejecutar un proceso de clasificación. En el título de la pestaña se indica el método que dio estos resultados. A la izquierda se muestran algunos de las características del clasificador más importantes, y a la derecha se encuentra la matriz de confusión que será la que muestre los resultados gráficamente.



Apéndice B

MANUAL DE REFERENCIA

La función de este anexo es proporcionar los detalles técnicos necesarios que permitan comprender el funcionamiento de la aplicación, así como la estructura de su código, por parte de cualquier desarrollador, para facilitar posibles trabajos adicionales de ampliación, modificación o continuación de este proyecto en un futuro, siguiendo las líneas ofrecidas en el apartado Trabajos Futuros de este documento.

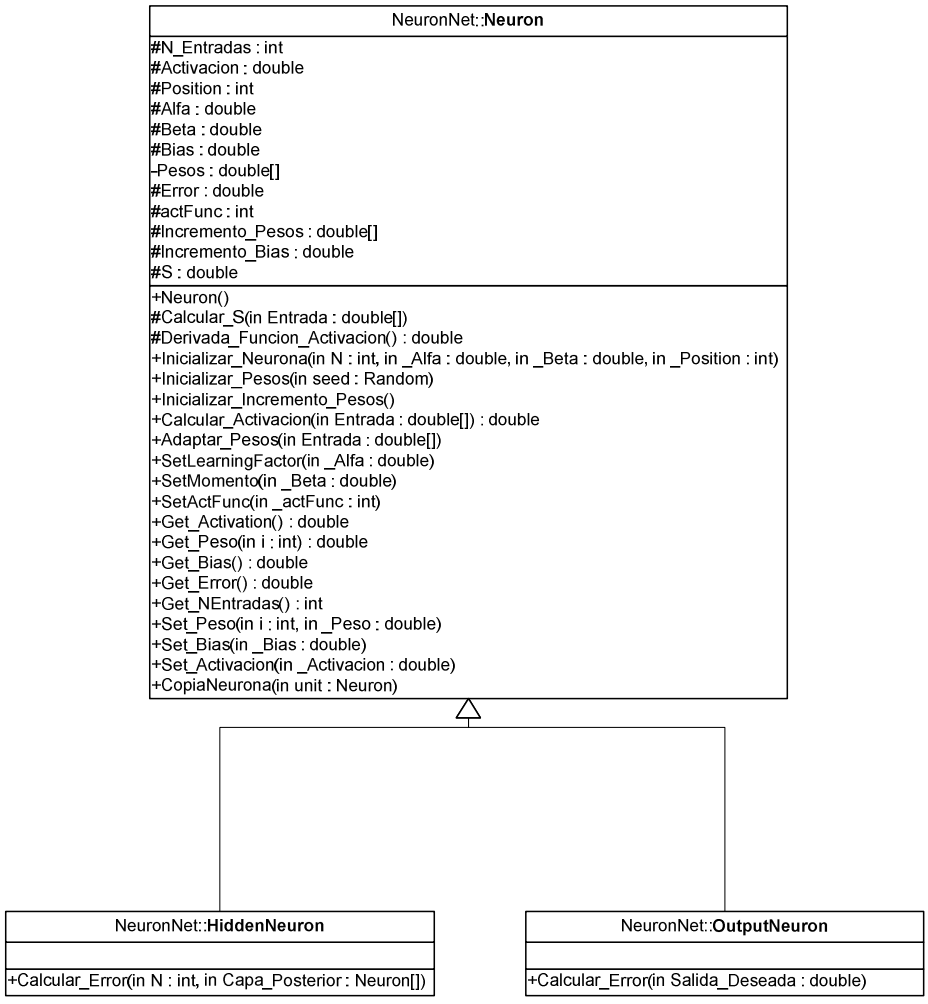
Para la implementación de esta aplicación se ha utilizado el lenguaje de programación orientado a objetos C# a través del entorno de desarrollo de Microsoft MS Visual Studio. Se ha considerado el uso de este lenguaje como la mejor opción debido a que permite un desarrollo sencillo y rápido y además proporciona una plataforma de desarrollo moderna, segura, abierta, robusta, viable y flexible.

A continuación se incluye una descripción detallada de las clases desarrolladas que forman parte de aplicación, así como información de los atributos y métodos más importantes en la implementación de los distintos puntos que comprende el trabajo realizado. Se presentan en un cuadro esquemático con la definición completa de la clase, y a continuación una tabla que contiene información de los atributos y métodos mencionados.

Primero se detallan las clases que implementan las Redes de Neuronas Artificiales, que comprenden las neuronas y el algoritmo de aprendizaje de retro-propagación.

- **NEURON, HIDDENNEURON y OUTPUTNEURON**

Las clases HIDDENNEURON y OUTPUTNEURON heredan de la clase NEURON, e implementan las neuronas ocultas y de salida. Contienen la información básica de las neuronas y los métodos que implementan los algoritmos de cálculo de sus propiedades.



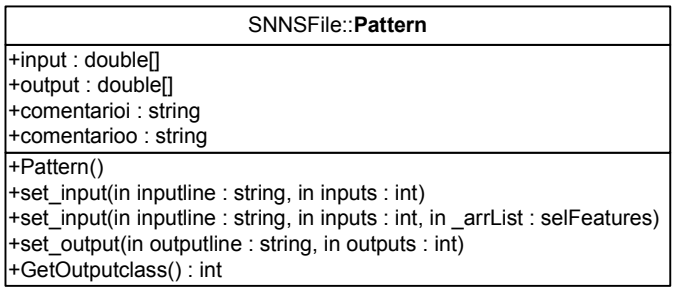
Clase NEURON	
Descripción	Clase que define una neurona genérica
Atributos más relevantes	- *N_Entradas* : número de entradas de la neurona - *Activacion* : valor de activación de la neurona - *Pesos* : vector con los valores del peso de los enlaces de la neurona - *S* : valor de la salida de la neurona
Métodos	- *Calcular_S* : Calcula y devuelve el valor de la salida de la neurona - *Inicializar_Neurona* : Inicializa la neurona con los valores felicitados - *Inicializar_Pesos* : Inicializa con valores aleatorios los pesos de la neurona - *Calcular_Activacion* : Calcula y devuelve el valor de activación de la neurona a partir de los datos de entrada - *Adaptar Pesos* : Modifica el valor de los pesos asignados a las entradas de la neurona - *Get_Activación* : Devuelve el valor de activación de la neurona - *Get_Peso* : Devuelve el valor del peso de la entrada que se indica en los parámetros de este método.

Clase HIDDENNEURON	
Descripción	Clase que define una neurona perteneciente a la capa oculta
Atributos más relevantes	*Atributos heredados de la clase NEURON*
Métodos	- *Calcular_error* : Calcula y devuelve el valor de la salida de la neurona de la capa oculta

Clase OUTPUTNEURON	
Descripción	Clase que define una neurona perteneciente a la capa de salida
Atributos más relevantes	*Atributos heredados de la clase NEURON*
Métodos	- *Calcular_error* : Calcula y devuelve el valor de la salida de la neurona de la capa de salida

- **PATTERN**

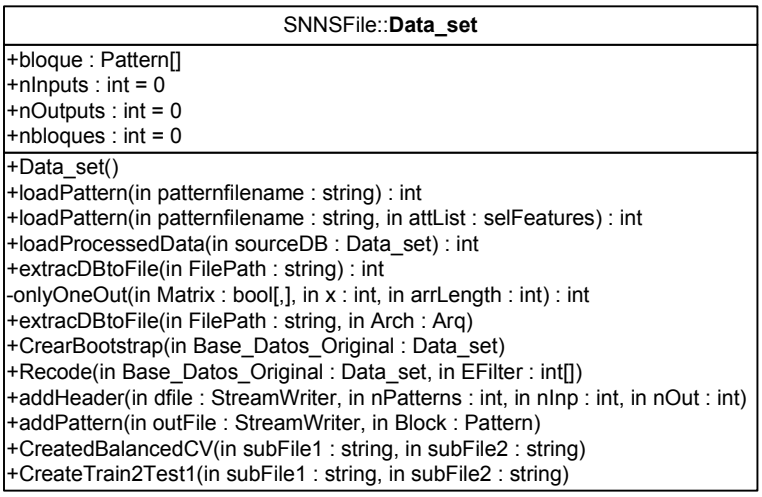
Implementa un ejemplo de entrada para una Red de Neuronas



Clase PATTERN	
Descripción	Clase que define un patrones de entrada o ejemplo
Atributos más relevantes	- *Input* : Array con los valores de los atributos del ejemplo *Output* : Array con los valores de las salidas del ejemplo. Contiene la información de la clase a la que pertenece el ejemplo
Métodos	- *Set_Input*: Carga los datos de entrada de un ejemplo a partir de una línea de fichero en un objeto de este tipo. - *Set_Output*: Carga los datos de salida un ejemplo a partir de una línea de fichero en un objeto de este tipo. - *GetOutputClass* : Devuelve la clase a la que pertenece el patrón

- **DATA_SET**

  Implementa un conjunto de datos de ejemplo provenientes de un fichero



Clase DATA_SET	
Descripción	Clase que define un conjunto de patrones de entrada
Atributos más relevantes	- *Bloque* : Vector con los patrones o ejemplos contenidos en el conjunto o fichero - *nInputs* : Número de características o atributos de entrada que tiene cada ejemplo - *nOutputs* : Número de clases que se definen en el dominio *nbloques* : Número de ejemplos que hay en el fichero
Métodos	- *Load_Pateinr* : Carga los datos de un fichero en un objeto de este tipo. - *Extract DB to file* : Almacena los datos del conjunto de ejemplos en un fichero - *CrearBootstrap* : Genera un *boot* o fichero con un subconjunto de datos, según el método explicado en *Baggin*. - *Recode* : Devuelve un fichero filtrado con la función correspondiente de la matriz ECOC que se le pasa como parámetro - *CreateBalancedCV* : Divide los ejemplos generando dos ficheros con el mismo número de ejemplos pertenecientes a cada clase en ambos - *CreateTrain2Test1* : Divide los ejemplos generando un fichero para entrenamiento de una red con 2/3 de los ejemplos y otro para test con los ejemplos restantes

- **BACKPROPAGATION**

Implementa una Red de Neuronas Artificiales con el algoritmo de retro-propagación como base para su aprendizaje.

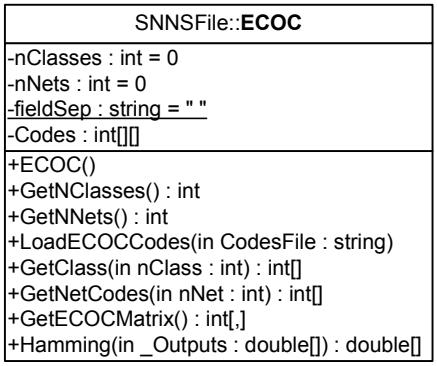
NeuronNet::**BackPropagation**
-Num_Entradas : int -Num_Neuronas_Ocultas : int -Num_Neuronas_Salida : int -Capa_Oculta : HiddenNeuron[] -Capa_Salida : OutputNeuron[] -Salidas_Capa_Oculta : double[] -Salida : double[] -Alfa : double -Beta : double -actFunc : int -SSE_Train : double -SSE_Test : double
+BackPropagation() +Create_Net(in N_Entradas : int, in N_Ocultas : int, in N_Salidas : int, in _Alfa : double) +Init(in R : Random) +Entrenamiento(in Datos : Data_set, in clase : int) +Entrenamiento(in Datos : Data_set) +Testear(in Datos : Data_set, in Aciertos : int) +Testear(in Datos : Data_set) +Testear(in Datos : Data_set, in Aciertos : int, in clase : int) +Analizar(in Entrada_Patron : double[], in Salida_Patron : double[]) : string +SetMomento(in _Beta : double) +SetActFunc(in type : char) : int +SetUpdateFunc(in function : string) +Get_SSE_Train() : double +Clear_SSE_Train() +Get_SSE_Test() : double +GetNumEntradas() : int +GetNumSalidas() : int +GetSalida(in i : int) : double +GetNumNeuronas(in Capa : int) : int +CopiaRed(in R : BackPropagation) +SetLearningFactor(in _Alfa : double) +Calcular_Salida_Red(in Patron : double[]) -Calcular_Activacion_Capa_Oculta(in Patron : double[]) -Calcular_Error_Cuadratico(in Salida_Esperada : double[]) : double +GuardarNet(in netfilename : string) : int +RecuperarNet(in netfilename : string) : int

Clase BACKPROPAGATION	
Descripción	Clase que define una Red de Neuronas Artificiales con el algoritmo de retro-propagación como base para su aprendizaje.
Atributos más relevantes	- *Num_Entradas* : número de neuronas de la capa de entrada de la Red - *Num_Neuronas_Ocultas* : número de neuronas de la capa oculta de la Red - *Num_Neuronas_Salida* : número de neuronas de la capa de salida de la Red - *Capa_Oculta* :array de objetos HIDDENNEURON, que contiene las neuronas de la

	capa oculta - *Capa_Salida* :array de objetos OUTPUTNEURON, que contiene las neuronas de la capa de salida de la Red - *Salidas_Capa_Oculta* : array de los valores de los enlaces de la capa Oculta y la de salida - *Salida* : array con los valores de las salidas de la Red
Métodos	- *Create_Net*: Crea la red de neuronas con los parámetros que se le indica - *Init*: Inicializa la Red con valores aleatorios - *Entrenamientos* : entrena la Red - *Testear* : Prueba la red con los ejemplos facilitados - *Calcular_Salida_Red* : Calcula la salida de la Red para un ejemplo de entrada - *Guardar Net* : Almacena los datos de la red en un fichero - *RecuperarNet* : Carga los datos de una Red desde un fichero

- **ECOC**

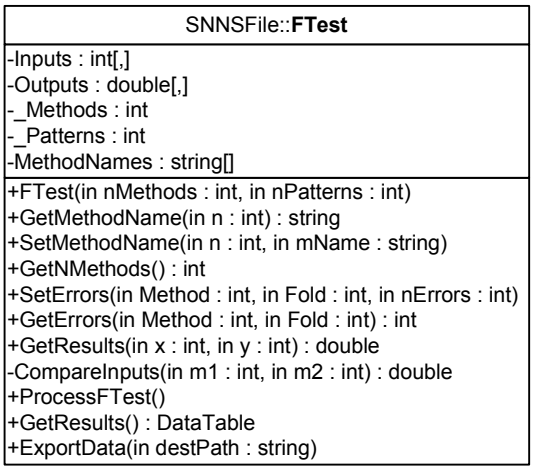
Implementa un objeto para almacenar la información de la matriz de códigos ECOC, con el algoritmo para calcular la distancia Hamming



Clase ECOC	
Descripción	Clase que define una matriz de código ECOC e implementa el cálculo de la distancia Hamming
Atributos más relevantes	- *nClasses* : Número de clases o filas de la matriz - *nNets* : Número de clases que se definen en el dominio *Codes* : matriz con los códigos
Métodos	- *LoadECOCCodes*: Carga los datos de un fichero en un objeto de este tipo. - *Hamming* : Dado un vector de valores, calcula y devuelve la distancia hamming a las clases de la matriz ECOC

- **F_TEST**

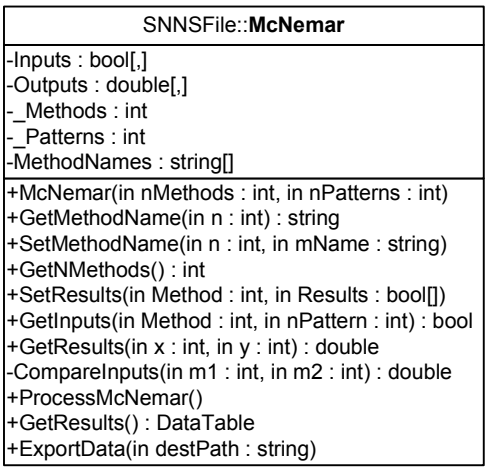
Implementa el algoritmo del cálculo del valor estadístico F_Test



Clase F_TEST	
Descripción	Clase que define la información resultante de un proceso de validación cruzada y calcula el valor de F_Test
Atributos más relevantes	- *Inputs* : Matriz con los resultados del proceso de validación cruzada. Sus valores representan el número de errores cometidos en cada iteración de la validación cruzada, correspondiendo cada columna a un método *Outputs* : Matriz con los valores resultantes de aplicar F_test a las comparaciones de los métodos implicados
Métodos	- *ProcessFTest* : Calcula los valores de F_Test para las comparaciones de todos los métodos, guardando los valores en la matriz de Outputs - *CompareInputs* : Calcula el valor de F_Test para los inputs que aportan dos métodos - *GetResults* : Devuelve una matriz con los valores de la matriz de Outputs - *ExportData* : Almacena los valores del objeto en un fichero

- **McNEMAR**

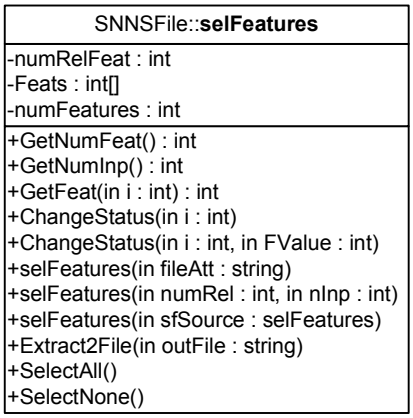
Implementa el algoritmo del cálculo del valor estadístico McNemar



Clase McNEMAR	
Descripción	Clase que define la información resultante de un proceso de pruebas y calcula el valor de McNemar
Atributos más relevantes	- *Inputs* : Matriz con los resultados del proceso de prueba de los distintos métodos. Contiene valores lógicos (true/false) para el resultado de clasificar cada ejemplo de prueba, por tanto tendrá tantas filas como ejemplos se hayan clasificado en la fase de test y tantas columnas como métodos se evalúan. *Outputs* : Matriz con los valores resultantes de aplicar McNemar a las comparaciones de los métodos implicados
Métodos	- *ProcessMcNemar* : Calcula los valores de McNemar para las comparaciones de todos los métodos, guardando los valores en la matriz de Outputs - *CompareInputs* : Calcula el valor de McNemar para los inputs que aportan dos métodos - *GetResults* : Devuelve una matriz con los valores de la matriz de Outputs - *SetResults* : Asigna los resultados de la fase de test de un método a una columna de la matriz de input - *ExportData* : Almacena los valores del objeto en un fichero

- **SELFEATURES**

Implementa un componente para la gestión de la selección de características



Clase SELFEATURES	
Descripción	Clase que define los datos involucrados en la selección de características
Atributos más relevantes	- *numRelFeat* : Número de atributos relevantes (seleccionados) - *Feats* : Lista de atributos con valor binario seleccionado/no seleccionado *numFeatures* : Número total de atributos
Métodos	- *ChangeStatus* : Asigna el estado a un atributo seleccionado/no seleccionado - *Extract2File* : Extrae los valores del obejto a un fichero de selección de atributos

- **CMATRIX**

Esta clase es la más importante de la aplicación, ya que contiene los atributos y métodos que implementan la construcción de los conjuntos de clasificadores. A través de esta clase, la aplicación gestiona los procesos de entrenamiento, prueba, etc. desarrollados, almacenando los resultados en una matriz de confusión

SNNSFile::**CMatrix**
-ConfMatrix : int[,] -_nOK : int -_nWrong : int -_nExamplesTrain : int -_nExamples : int -_nExamplesTest : int -_nFeatures : int -_nClasses : int -_Method : string -featSelection : bool -featSelected : selFeatures -_nBoots : int -_nCycles : int -_nHiddenNeurons : int -_SSE_End : double -_LearningFactor : double -_ECOC : ECOC -_TestResults : bool[]
+LoadECOC(in ECOCFile : string) +GetECOC() : ECOC +CMatrix(in intMethod : int, in Boots : int, in Cycles : int, in Hidden : int, in SSEE : double, in LFactor : double) +CMatrix(in _Source : CMatrix) +GetTestResults() : bool[] +GetTestResults(in i : int) : bool +SetTestOK(in i : int) +SetTestWrong(in i : int) +SetFeatSelection(in featList : selFeatures) +SetClasses(in _classes : int) +GetClasses() : int +GetMatrix() : int[,] +GetDTMatrix() : DataTable +GetDTProperties() : string[] +ExportData(in filePath : string) +Process(in TrainingFile : FileInfo, in WorkingPath : string, in EsbFileName : string) : int +Process(in EsbFileName : string, in TestFile : FileInfo, in WorkingPath : string) : int +Process(in TrainingFile : FileInfo, in TestFile : FileInfo, in WorkingPath : string) : int +Process(in TrainingFile : FileInfo, in TestFile : FileInfo, in WorkingPath : string, in EsbFileName : string, in Train : bool, in Test : bool) : string -SimpleNet(in TrainFile : FileInfo, in WorkingPath : string, in EnsembleFileName : string) -SimpleNet(in TrainFile : FileInfo, in TestFile : FileInfo, in WorkingPath : string) -SimpleNet(in EnsembleFileName : string, in TestFile : FileInfo, in WorkingPath : string) -Bagging(in TrainFile : FileInfo, in TestFile : FileInfo, in WorkingPath : string) -Bagging(in TrainFile : FileInfo, in WorkingPath : string, in EnsembleFileName : string) -Bagging(in EnsembleFileName : string, in TestFile : FileInfo, in WorkingPath : string) -OAA(in TrainFile : FileInfo, in WorkingPath : string, in EnsembleFileName : string) -OAA(in TrainFile : FileInfo, in TestFile : FileInfo, in WorkingPath : string) -OAA(in EnsembleFileName : string, in TestFile : FileInfo, in WorkingPath : string) -OAA(in TrainFile : FileInfo, in TestFile : FileInfo, in WorkingPath : string, in EnsembleFileName : string, in Train : bool, in Test : bool) : string -ECOC(in TrainFile : FileInfo, in TestFile : FileInfo, in WorkingPath : string) -ECOC(in TrainFile : FileInfo, in WorkingPath : string, in EnsembleFileName : string) -ECOC(in EnsembleFileName : string, in TestFile : FileInfo, in WorkingPath : string) +nOK() : int +nWrong() : int +nExamples() : int +nClasses() : int +Method() : string +nBoots() : int +nCycles() : int +nHiddenNeurons() : int +SSE_End() : double +LearningFactor() : double

Clase SELFEATURES	
Descripción	Clase que define los datos involucrados en la selección de características
Atributos más relevantes	- *ConMatrix* : Matriz de confusión con los valores resultantes de un proceso de test, con los métodos implementados - *_OK* : Número total de aciertos del proceso de test - *_nWrong* : Número total de errores del proceso de test - *_nExamplesTest* : Número de ejemplos empleados para entrenar las Redes - *_nExamplesTest* : Número de ejemplos empleados en la fase de test - *_Method* : Método utilizado en la construcción del Conjunto de Clasificadores - *_nFeatures* : Número de características de las entradas de cada ejemplo - *featSelected* : Objeto selFeatures con la información de la selección de características - *_nBoots* : Número de Boots que se emplean en el proceso Bagging - *_nClasses* : Número de clases del dominio - *_nCycles* : Número de ciclos de aprendizaje empleados en el entrenamiento - *_nHiddenNeurons* : Número de neuronas de la capa oculta - *_LearningFactor* : Valor del factor de aprendizaje - *_ECOC* : Objeto ECOC con los datos de la matriz de códigos en el proceso ECOC *_TestREsults* : Array con los resultados por ejemplo del proceso de prueba
Métodos	- *LoadECOC* : Carga la matriz de códigos ECOC del fichero especificado en el objeto ECOC del objeto - *SetTestOK, SetTestWrong* : Asigna acierto o error en el array testresults a un ejemplo clasificado - *SetFeatSelection* : Carga los datos de selección de características a partir de un fichero especificado - *GetDTMatrix* : Devuelve los resultados de la matriz de confusión en un objeto de tipo datatable - *ExportData* : Guarda los datos de la matriz de confusión en un fichero especificado - *Process* : Crea y entrena, o prueba, o ambas cosas, un Conjunto de Clasificadores según el método especificado - *SimpleNet* : Crea y entrena, o prueba, o ambas cosas, un Conjunto de Clasificadores según el método SimpleNet - *Bagging* : Crea y entrena, o prueba, o ambas cosas, un Conjunto de Clasificadores según el método Bagging - *OAA* : Crea y entrena, o prueba, o ambas cosas, un Conjunto de Clasificadores según el método OAA - *ECOC* : Crea y entrena, o prueba, o ambas cosas, un Conjunto de Clasificadores según el método ECOC